المملكـــة العربيـــة الســـعودية

المؤسسة العامة للتعليم الفني والتدريب المهني

الإدارة العامة لتصميم وتطوير المناهج



البرمجيات

برمجة ٣

٢٤٣ حاب

مقدمة

الحمد لله وحده، والصلاة والسلام على من لا نبي بعده، محمد وعلى آله وصحبه، وبعد:

تسعى المؤسسة العامة للتعليم الفني والتدريب المهني لتأهيل الكوادر الوطنية المدربة القادرة على شغل الوظائف التقنية والفنية والمهنية المتوفرة في سوق العمل، ويأتي هذا الاهتمام نتيجة للتوجهات السديدة من لدن قادة هذا الوطن التي تصب في مجملها نحو إيجاد وطن متكامل يعتمد ذاتياً على موارده وعلى قوة شبابه المسلح بالعلم والإيمان من أجل الاستمرار قدماً في دفع عجلة التقدم التنموي: لتصل بعون الله تعإلى لمصاف الدول المتقدمة صناعياً.

وقد خطت الإدارة العامة لتصميم وتطوير المناهج خطوة إيجابية تتفق مع التجارب الدولية المتقدمة في بناء البرامج التدريبية، وفق أساليب علمية حديثة تحاكي متطلبات سوق العمل بكافة تخصصاته لتلبي متطلباته ، وقد تمثلت هذه الخطوة في مشروع إعداد المعايير المهنية الوطنية الذي يمثل الركيزة الأساسية في بناء البرامج التدريبية، إذ تعتمد المعايير في بنائها على تشكيل لجان تخصصية تمثل سوق العمل والمؤسسة العامة للتعليم الفني والتدريب المهني بحيث تتوافق الرؤية العلمية مع الواقع العملي الذي تفرضه متطلبات سوق العمل، لتخرج هذه اللجان في النهاية بنظرة متكاملة لبرنامج تدريبي أكثر التصاقاً بسوق العمل، وأكثر واقعية في تحقيق متطلباته الأساسية.

وتتنأول هذه الحقيبة التدريبية " برمجة ٣ " لمتدربي قسم" البرمجيات " للكليات التقنية موضوعات حيوية تتنأول كيفية اكتساب المهارات اللازمة لهذا التخصص.

والإدارة العامة لتصميم وتطوير المناهج وهي تضع بين يديك هذه الحقيبة التدريبية تأمل من الله عز وجل أن تسهم بشكل مباشر في تأصيل المهارات الضرورية اللازمة، بأسلوب مبسط يخلو من التعقيد، وبالاستعانة بالتطبيقات والأشكال التي تدعم عملية اكتساب هذه المهارات.

والله نسأل أن يوفق القائمين على إعدادها والمستفيدين منها لما يحبه ويرضاه إنه سميع مجيب الدعاء.

الإدارة العامة لتصميم وتطوير المناهج

تمهيد



برمجة ٣

الوراثة و تعدد الأشكال

الوراثة و تعدد الأشكال

١

الجداره:

أن يكون المتدرب قادراً على فهم المبادىء الأساسية لبرمجة الكائنات مثل الوراثة وتعدد الأشكال والقدره على كتابة برامج تحتوي على فصائل جديدة ترث طرق وبيانات فصائل تم إنشاؤها وتنفيذها بصورة جيدة.

الأهـداف:

١. مراجعة المفاهيم الأساسية لبرمجة الكائنات في لغة الجافا
٢. تعلم مبادىء الوراثة
٣. فهم كيفية وراثة واستبدال طرق الفصائل العليا
٤. فهم فكرة تعدد الأشكال

مستوى الاداء المطلوب:

أن يصل المتدرب إلى اتقان الجدارة بنسبة ١٠٠٪

الوسائل المساعده:

- وجود حاسب آلي
- وجود بيئة متكاملة لبناء وتنفيذ برامج لغة الجافا
- دفتر
- قلم

الوراثة وتعدد الأشكال

Inheritance and Polymorphism

مقدمة

مقرر برمجة ٢ تتأول المفاهيم الأساسية لبرمجة الكائنات Object Oriented programming وأن لغة الجافا تعتمد على هذه المفاهيم وفي هذه المقدمة نختصر بعض هذه المفاهيم لتكون اساساً للموضوعات المتقدمة في مجال برمجة الكائنات التي يتتأولها هذا المقرر.

من دراستنا السابقة للغة الجافا تبين أن البرنامج يحتوي على مجموعة من الفصائل Classes ويمكن إنشاء مجموعة من الكائنات Objects التي تأخذ خصائص الفصيلة المنشأة منها وأن كل فصيلة تحتوي على مجموعة من الطرق Methods التي تبين سلوك الكائنات وكذلك تحتوي الفصيلة على مجموعة من المتغيرات Instance variables التي تحتفظ بخصائص الكائن المنشئا من هذه الفصيلة ويبين شكل(١ - ١) مثال لبناء فصيلة لحساب البنك وسوف نقوم بشرح مكونات هذه الفصيلة لمراجعة المفاهيم الأساسية لبرمجة الكائنات في لغة الجافا وتكون أساساً لبناء المفاهيم المتطورة.

أولاً الفصيلة Class

الفصيلة هي القالب الذى نستخدمه في إنشاء الكائنات Objects وكل فصيلة لها خصائصها Attributes والتي تحددها البيانات Data أو المتغيرات Member variables ولها سلوكها Behaviorوهي الطرق Member Methods والشكل العام لتعريف فصيلة هو:

```
Access specifier        class             class_name
Ex:
public            class           BankAccount □
```

عند إنشاء برنامج بلغة الجافا يحتوي على العديد من الفصائل يمكن وضع كل فصيلة في ملف منفصل ويبداء بمحدد الوصول للفصيلة عام public أو تجميع الفصائل في ملف واحد ويبدأ تعريف الفصيلة التي تحتوي على الطريقة main بكلمة الوصول عام public class وتبدأ باقىالفصائل بالكلمة المحجوزة class فقط.

ثانياً المنشآت Constructors

تحتوي هذه الفصيلة على نوعين من المنشآت Constructors

١. المنشئا الأول وهو الإفتراضي ويعطى قيمة صفر للحساب أثناء إنشاء كائن جديد من هذه الفصيلة وهذا المنشأ لايحتوي اي عوامل Parameters داخل الأقواس ولكن تظل الأقواس مطلوبة

BankAccount.java

```
1  public class BankAccount
2  {
3  // The first constructor is the default constructor sets balance to zero
4          public BankAccount()
5          {
6                  balance = 0;
7          }
8  // The second constructor sets balance to initial to initial balance
9          public BankAccount(double initialBalance)
10         {
11                 balance = initialBalance;
12         }
13  // The deposit method adds an amount to instance variable balance
14         public void deposit(double amount)
15         {
16                 balance = balance + amount;
17         }
18  // The withdraw method subtracts an amount from instance variable balance
19         public void withdraw(double amount)
20         {
21                 balance = balance - amount;
22         }
23  // The transfer method withdraw an amount from this object and deposit to
other object balance
24         public void transfer(BankAccount other , double amount)
25         {
26                 withdraw(amount);
27                 other.deposit(amount);
28         }
29
30         // The getBalance method returns the current balance
31         public double getBalance()
32         {
33                 return balance;
34         }
35                 // The instance variable balance
36         private double balance;
37 }
```

شكل (١ -١)

٢. المنشأ الثانى ويعطى قيمة ابتدائية للحساب أثناء إنشاء كائن جديد من هذه الفصيلة ويحتوي بين الأقواس على عامل واحد من نوع البيانات الرقمية ذي الفصلة العشرية double

ومن الملاحظ في هذا المثال وفي لغة الجافا بصفة عامة أن

١. الغرض من المنشآت Constructors هو إعطاء قيم أولية لمتغيرات الحالة للكائن عند إنشائه أول مرة من الفصيلة
٢. المنشآت تأخذ نفس اسم الفصيلة
٣. الفصيلة يمكن أن تحتوي على العديد من المنشآت ويقوم المترجم بتحديد أي منهم يستدعى من خلال العوامل داخل الأقواس
٤. تكون الصيغة العامة للمنشأ في لغة الجافا كالتالي

```
Access Specifier      Class name (parameter type    parameter name, ….)
Ex:
Public                BankAccount (double           initialBalance)□
```

ثالثاً الطرق Methods

الفصيلة BankAccount تتكون من أربعة طرق Methods تمثل العمليات الأساسية في تعامل البنوك مع العملاء وهي

١ - عملية الإيداع deposite Method
٢ - عملية السحب withdraw method
٣ - عملية الاستعلام عن الرصيد getBalance method
٤ - عملية التحويل من حساب إلى حساب transfer method

الشكل العام لعنوان الطريقة method هو :

```
Access specifier      return type      method name(parameter type parameter name,..)
Ex 1:
public          void             deposite (double amount)
Ex 2:
public          double           getBalance
```

- المجال الأول في العنوان يبين محدد الوصول إلى الطريقة عام public
- المجال الثاني في العنوان يبين نوع البيانات العائدة بعد التنفيذ مثل double ولتبين أن الطريقة لاتسترجع أي بيانات نستخدم الكلمة void.

- المجال الثالث وهو اسم الطريقة وهو إختياري ومن الأفضل استخدام اسم يدل على وظيفة الطريقة ودائما في لغة الجافا يبدأ الاسم بحرف صغير small letter وفي حالة الاسم الذى يتكون من أكثر من مقطع يبدأ المقطع الأول بحرف صغير ثم المقاطع الآخرى بحرف كبير مثل getBalance
- المجال الرابع وهو الأقواس وفي داخلها معاملات الطريقة يفصل بينهما فصلة (,) وفي حالة عدم وجود عوامل تظل الأقواس مستخدمة وهذه العوامل تسمى عوامل ظاهرة explicit parameter
- هناك نوع آخر من العوامل يخص الطريقة وهو غير الظاهر(الضمني) implicit parameters هذا العامل الضمني من نوع الفصيلة التي تعرف الطريقة

رابعاً الكائنات Objects **وأحيانا تسمى** Instance

وهي عناصر تمثيل استخدام الفصيلة في البرنامج وهي تأخذ نفس شخصية الفصيلة من دوال وبيانات والطريقة الوحيدة لإنشاء الكائن بإستخدام المؤثر new

Ex1:
```
BankAccount myAccount = new BankAccount();
```

الجملة السابقة في لغة الجافا تقوم بإنشاء كائن يسمى myAccount ويستدعى المنشيء الأول الإفتراضي BankAccount() في الفصيلة BankAccount ويعطى قيمة أولية صفر لنسخة متغير الكائن balance للكائن myAccount

Ex2:
```
BankAccount m1 = new BankAccount(5000);
```

هذه الجملة في لغة الجافا تقوم بإنشاء كائن يسمى m1 وتستدعي المنشيء الثاني BankAccount(double initialBalance) من الفصيلة BankAccount ويعطى قيمة أولية ٥٠٠٠ ريال لنسخة متغير الكائن balance للكائن m1

خامساً متغيرات الكائنات instance variables

كما رأينا في الأمثلة السابقة لإنشاء الكائنات أن كل كائن يخزن حالته في واحد أو أكثر من متغيرات الكائن ويكون الشكل العام لتعريف متغير الكائن هو:

```
Access specifier       type       variable name;
Ex:
private            double        balance;
```

المجال الأول ويعرف بمحدد الوصول للمتغير وبصفة عامة نستخدم المحدد خاص private لمتغيرات الكائنات وهذا يعني أنه يمكن الوصول لهذه المتغيرات فقط بدوال نفس الفصيلة المعرف فيها المتغير ولايمكن تغييره من أى دالة آخرى ولذلك نستطيع القول بأن متغير الكائن غير ظاهر للمبرمج الذى يستخدم الفصيلة المعرف فيها وهذه العملية من إخفاء البيانات data hiding تسمى تغليف Encapsulation وهي المبدأ الأول من مبادىء برمجة الكائنات OOP

ونلاحظ أن كل كائن له نسخته من المتغيرات

```
Ex1:
myAccount.balance
Ex2:
m1.balance
```

فعندما تشير إلى متغير فيطريقة فإنك تشير تلقائياً إلى متغير الكائن المستخدم في استدعاء الطريقة وفي هذه الحالة يكون الكائن هو المتغير الضمنى للطريقة

```
Ex:
m1.deposit(500);
```

عند تنفيذ هذه الجملة في لغة الجافا تقوم باستدعاء الطريقة deposit وتضيف ٥٠٠ ريال إلى حساب الكائن m1 وكأنه ينفذ الجملة في الطريقة deposit كالتإلي

```
balance = balance + amount
this.balnce = this.balance + amount
m1.balance = m1.balance +amount
```

الوراثة Inheritance

الوراثة inheritance هي المبدىء الثاني من مبادىء برمجة الكائنات OOP والتي يمكن الإستفادة منها في لغة جافا لتطوير البرامج حيث يمكن استخدام الفصائل التي تم تصميمها وتنفيذها وتأكدنا من أنها تعمل بصورة جيدة ثم نكتب فصيلة جديدة يضاف اليها الطرق والبيانات الجديدة فقط وترث الطرق والبيانات الموجودة في الفصيلة القديمة التي يمكن اعتبار الفصيلة الجديدة امتداداً لها

مثال : نفترض أننا نريد إنشاء حساب بنكي يضيف عائداً شهرياً على الرصيد الموجود في البنك.

بدراسة هذا النوع من الحساب يتضح لنا أن هذا الحساب هو امتداد لحساب البنك التي تم برمجته في الفصيلة BankAccount حيث إنه يتطلب عمليات ايداع وسحب واستعلام عن رصيد وله متغير لتخزين نسخة من قيمة الحساب ولذلك يمكن استخدام مبدأ الوراثة لتطوير البرنامج السابق بإنشاء فصيلة جديدة ويضاف إليها دالة لحساب العائد ويضاف إليها أيضا متغير لمعدل العائد الشهري. الشكل (١ -٢) يبين بناء هذه الفصيلة.

SavingAccount.java

```
/* The SavingAccount class extends the BankAccount class implements
   a new method addInterest to model an account that pays a fixed
   interest rate on deposits
*/
public class SavingAccount extends BankAccount
// The SavingAccount constructor
{
        public SavingAccount(double rate)
        {
                interestRate = rate;
        }
// addInterest method
        public void addInterest()
        {
                double interest = getBalance()*interestRate/100;
                deposit(interest);
        }
// The SavingAccount instance variable
        private double interestRate;
}
```

شكل (١ -٢)

وبتحليل هذا المثال يمكن أن نستعرض بعض خصائص الوراثة في لغة جافا ومنها

١ - من أسباب استخدام الوراثة هو إعادة استخدام شفرة البرنامج code reuse حيث يمكن استخدام فصائل موجودة ونوفر الجهد المبذول لإتقان تصميم وتنفيذ هذه الفصائل.

٢ - الفصيلة التي تورث تسمى الفصيلة العامة (العليا) superclass لأنها تحتوي على الطرق والبيانات المشتركة وأحيانا تسمى فصيلة الأب parent class أو الفصيلة الأساسية Base class

٣ - الفصيلة التي ترث تسمى الفصيلة الفرعية subclass لأنها تحتوي على الطرق والبيانات الخاصة المضافة وأحيانا تسمى فصيلة الابن child class أو الفصيلة المشتقة drived class

٤ - ولتحقيق عملية الوراثة وتوريث فصيلة قديمة إلى فصيلة جديدة عند إنشائها نقوم بكتابة اسم الفصيلة الجديدة ثم الكلمة المحجوزة الدالة على الوراثة extends التي تعني أن هذه الفصيلة هي امتداد للفصيلة القديمة التي يكتب اسمها بعدها وتكون الصيغة العامة للوراثة هي:

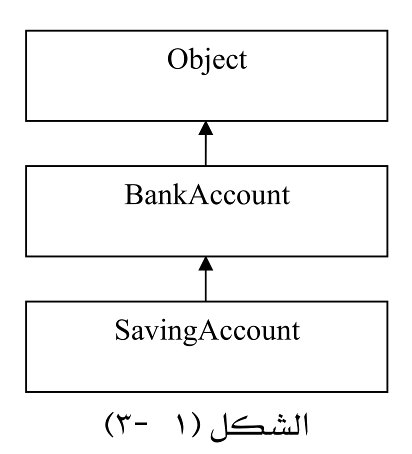
```
class subclass Name        extends        superclass Name
ex:
class SavingAccount        extends        BankAccount
```

وهذا يعني أن في المثال السابق الفصيلة BankAccount هي الفصيلة العامة (العليا) superclass وأن الفصيلة SavingAcount هي الفصيلة الفرعية subclass

٥ - عندما نقوم بإنشاء فصيلة ولم نحدد اسم فصيلة ترث منها تفترض لغة جافا أنك ترث من الفصيلة الأم Object ومثال ذلك الفصيلة BankAccount ترث الفصيلة Object

٦ - الفصيلة Object تحتوي على عدد صغير من الطرق التي تعني شيئاً لجميع الكائنات مثل الطريقة toString التي يمكن استخدامها للحصول على وصف حالة الكائن.

الشكل (١ -٣) يبين علاقة الوراثة بين الفصائل الثلاثة Object و BankAccount و SavingAccount و يسمى مخطط الوراثة Inheritance Diagram



الشكل (١ -٣)

٧ - عند إنشاء كائن جديد من الفصيلة الفرعية ينادى أولاً المنشيء الموجود في الفصيلة العامة ليعطي قيمة مبدئية للمتغيرات الموجودة فيها ثم ينفذ المنشيء الموجود في الفصيلة الفرعية لإعطاء قيم أولية للمتغيرات الجديدة

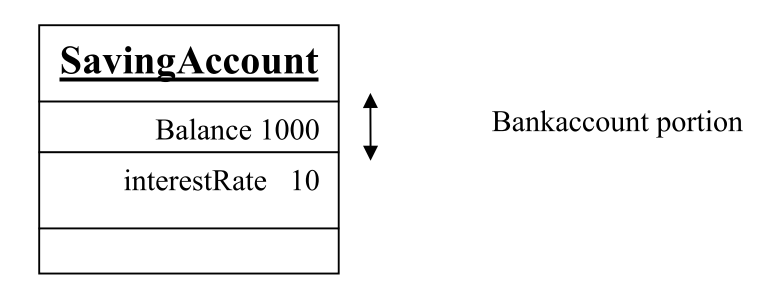
Ex:

```
SavingAccount          m2 = new SavingAccount(10)
```

هذه الجملة في لغة الجافا تقوم بإنشاء كائن m2 من نوع حساب البنك SavingAccount وتستدعي المنشيء الافتراضي في الفصيلة العامة BankAccount لإعطاء قيمة صفر للمتغير balance ثم تستدعى المنشيء الموجود في الفصيلة Saving Account لإعطاء قيمة أولية ١٠ للمتغير intrestRate وبذلك تكون نسخة متغيرات الكائن بعد تنفيذ هذه الجملة كالأتي :

```
m2.balance = 0
m2.intrestRate = 10
```

أي أن الكائن من الفصيلة الفرعية ورث المتغير balance من الفصيلة العامة وأضيف إليه المتغير intrestRate من فصيلته ويبين الشكل (١ -٤) مثال للكائن m2 من الفصيلة SavingAccount



شكل (١ -٤)

٨ - الكائن المنشيء من الفصيلة الفرعية يعتبر حالة خاصة من الفصيلة العامة ويمكنه استدعاء الطرق الموجودة فيها كما ينادي الطرق الموجودة في الفصيلة الفرعية.

Ex: m2.addIntrest();

هذه الجملة في لغة جافا تستدعي الطريقة addIntrest من الفصيلة الفرعية SavingAccount التي يتم فيها حساب العائد على الحساب الحالي وإضافته على الحساب وتلاحظ في هذه الطريقة أنها تستدعي الطرق getBalance و deposit من الفصيلة العامة BankAccount لأن الفصيلة الفرعية ترث الطرق من الفصيلة العامة وبسبب عدم وجود كائن أثناء استدعاء هذه الطرق فإنها تستخدم المتغير الضمني الذي يستدعي الطريقة addIntrest وتنفذ جمل الطريقة addIntrest كالآتى :

```
double   interest =this.getBalance() * this.intrestRate/100
this.deposit(intrest)
```

وبتنفيذ الجملة ;()m2.addIntrest يكون التنفيذ لجمل الطريقة ()addIntrest كالأتى:

```
double   interest =  m2.getBalance() * m2.intrestRate/100
m2.deposit(intrest)
```

٩ - يجب أن نعلم أن أي فصيلة لها فصيلة عليا واحدة فقط ولكن الفصيلة العليا يمكن أن يكون لها أي عدد من الفصائل الفرعية ولذلك يطلق على لغة جافا أنها تستخدم single inheritance وذلك عكس لغة ++C التي يمكن فيها أن يكون لها أكثر من فصيلة عليا ويطلق على هذا النوع من الوراثة Multiple inheritance وعلى الرغم من أن هذا النوع يمثل قوة في البرمجة إلا أنه يسبب الكثير في التعقيدات من التصميم وتتبع هيكل الفصائل.

المخطط الهرمي للوراثة Inheritance Hierarchies

المخطط الهرمي عادةً يمثل كشجرة حيث معظم الفصائل التي تمثل المفاهيم العامة قريبة من الجزر root والفصائل الأكثر تخصصية نحو الفروعbranches وسوف نستخدم مثال بسيط للمخطط الهرمي لإستكمال دراسة مفاهيم الوراثة.

المثال : نفترض أن البنك يريد أن يقدم لعملائه ثلاثة أنواع من الحسابات:

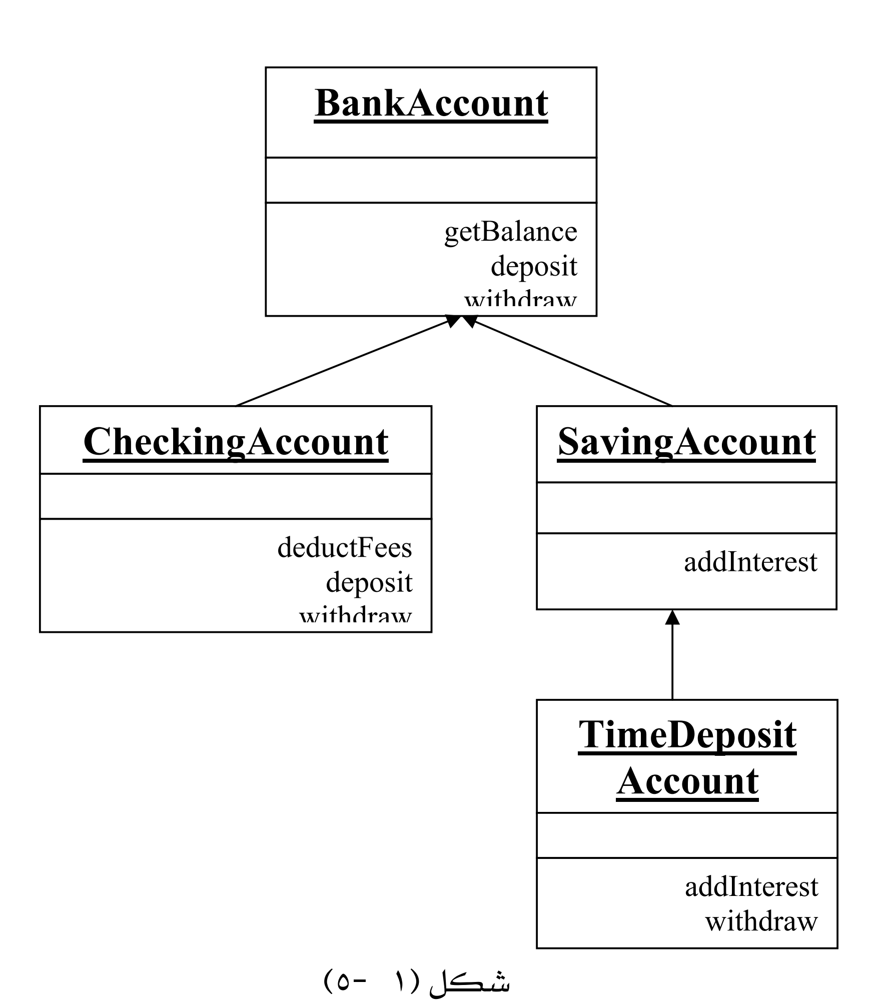
١ - الحساب الأول CheckingAccount ليس له عائد ويسمح للعميل بعدد قليل من العمليات البنكية كل شهر ثم يلزمه بدفع أجرة عن كل عملية إضافية.

٢ - الحساب الثاني SavingAccount يعطي عائداً مركباً شهرياً

٣ - الحساب الثالث TimeDepositAccount يعطى عائد مركب شهرياً ولكن يلتزم العميل بترك المال في الحساب لعدد معين من الشهور بدون سحب ويوجد شرط جزائي للسحب المبكر.

بتحليل هذه الحسابات نجد أنها حسابات بنكية تشترك جميعها في المتغير balance وكذلك في عمليات الإيداع deposit والسحب withdraw و االاستعلام عن الرصيد getBalance ولذلك يمكن القول أنها جميعاً

يمكن أن ترث الفصيلة BankAccount وكذلك يمكن القول بأن الحساب الثاني والثالث متشابهان ومختلفان عن الحساب الأول ولذلك يمكن تمثيل المخطط الهرمي للوراثة لهذه الفصائل البنكية كما هو موضح في الشكل (١ -٥)



شكل (١ -٥)

الطرق methods ومتغيرات الكائنات instance variables للفصائل الفرعية

أولاً الطرق Methods :

عند تعريف طرق فصيلة فرعية يمكن أن يكون هناك ثلآثة احتمالات

١ - استخدام الطريقة في الفصيلة الفرعية بنفس الاسم ونفس المعاملات (أي نفس البصمة) كما في الفصيلة العامة ولكن هناك استبدال للجمل التنفيذية لهذه الطريقة وتسمى override method ولذلك عند استدعاء هذه الطريقة باستخدام متغير من نوع الفصيلة الفرعية فيتم تنفيذ الطريقة الموجودة في الفصيلة الفرعية وليست الطريقة الأصلية الموجودة في الفصيلة العليا.

٢ - يمكن للفصيلة الفرعية أن ترث الطرق الموجودة في الفصيلة العليا بدون أى تغير وفي هذه الحالة عند استدعاء هذه الطرق باستخدام كائن من نوع الفصيلة الفرعية يتم تنفيذ الطريقة من الفصيلة العليا.

٣ - يمكن للفصيلة الفرعية إنشاء طرق جديدة لتحقيق الغرض من التطوير وتستدعى هذه الطرق الجديدة فقط باستخدام كائنات الفصيلة الفرعية.

ثانياً متغيرات الكائنات Instance variables

بالنسبة لمتغيرات الكائنات يوجد حالتان فقط

١ - جميع متغيرات الكائنات للفصيلة الفرعية تورث تلقائياً لكائنات الفصيلة الفرعية

٢ - يمكن تعريف متغير جديد يطبق فقط على كائنات الفصيلة الفرعية

ويمكن تطبيق هذه الاحتمالات على الحساب الأول فقد تم إنشاء فصيلة جديدة تسمى CheckingAccount كما في شكل (١ -٦) وهذه الفصيلة ترث فصيلة BankAccount وفيها تم إضافة طريقة جديدة deductFees() لحساب الأجرة الشهرية وأيضا تم إضافة متغير كائن transactionCount لتتبع عدد العمليات الشهرية ونجد أيضاً أنه تم استبدال الطرق deposit و withdraw لزيادة أعداد العمليات transactionCount أثناء عمليات السحب ولايداع

الكائن المنشىء من الفصيلة CheckingAccount يكون له متغيران

١ - الأول balance يرثه من الفصيلة BankAccount

٢ - الثاني transactionCount معرف جديد في فصيلة CheckingAccount

ويطبق على الكائن المنشىء من الفصيلة CheckingAccount أربعة طرق وهي :

١ - الاستعلام عن الرصيد getBalance() يرثها من فصيلة BankAccount

٢ - الإيداع deposit(double amount) (تستبدل الطريقة الموجودة في الفصيلة BankAccount)

٣ - السحب withdraw(double amount) (تستبدل الطريقة الموجودة في الفصيلة BankAccount

٤ - الطريقة deductFees() معرفة جديدة في فصيلة CheckingAccount

والآن دعنا نشرح تنفيذ دالة الإيداع deposit(double amount) في الفصيلة CheckingAccount لنبين كيفية تغير متغير كائن في الفصيلة العليا وكيفية استدعاء دالة من الفصيلة العليا لها نفس البصمة في الفصيلة الفرعية.

CheckingAccount.java

```
/* The CheckingAccount class extends the BankAccount class implements
   a new method deductFees and overide the deposit and withdraw methods
*/
public class CheckingAccount extends BankAccount
// The CheckingAccount constructor
{
            public CheckingAccount(double initialBalance)
            {
                        // construct superclass
                        super(initialBalance);
                        // initialize transaction count
                         transactionCount = 0;
            }
// overide the BankAccount deposit method
            public void deposit(double amount)
            {
                        transactionCount ++;
                        // now add amount to balance
                        super.deposit(amount);
            }
// overide the BankAccount withdraw method
            public void withdraw(double amount)
            {
                        transactionCount ++;
                        // now subtract amount from balance
                        super.withdraw(amount);
            }
// New method deductFees
            public void deductFees()
            {
                        if(transactionCount>FREE_TRANSACTIONS)
                                    {           double fees =
TRANSACTION_FEE*(transactionCount- FREE_TRANSACTIONS);
                                                super.withdraw(fees);
                                    }
                        transactionCount = 0;

            }
// The CheckingAccount instance variables
            private int transactionCount;
            private static final int FREE_TRANSACTIONS = 3;
            private static final double TRANSACTION_FEE = 2.0;
}
```

شكل (١ -٦)

طريقة الإيداع في الفصيلة الفرعية هي مستبدلة override من الفصيلة العليا فلها نفس الاسم والعوامل والغرض منها هو زيادة أعداد العمليات وإيداع المبلغ فتكون بالصورة الآتية

```
// override the BankAccount deposit method
        public void deposit(double amount)
        {
                transactionCount ++;
                // now add amount to balance
                balance = balance + amount;          //ERROR
        }
```

تنفيذ الإيداع بهذه الطريقة خطأ لأن محدد الوصول للمتغير balance معرف في الفصيلة العليا أنه خاص private ولذلك فإن دوال الفصيلة الفرعية ليست لها الأحقية في تغيير البيانات الخاصة في الفصيلة العليا ولذلك يجب استخدام دوال الفصيلة العليا لتعديل متغيرات الكائن فيها فإذا استخدمنا الطريقة deposit(amount) من الفصيلة العامة لتغيير قيمة المتغير balance يصبح بناء الطريقة في الفصيلة الفرعية كالآتى

```
// override the BankAccount deposit method
        public void deposit(double amount)

                transactionCount ++;
                // now add amount to balance
                deposit(amount);
```

هنا نجد مشكلة آخرى وهي أن الطريقة deposit(amount) تستدعى بالمتغير الضمني this وهو من نوع الفصيلة CheckingAccount التي تملك دالة بنفس البصمة deposit(double amount) ولذلك فإن تنفيذ البرنامج يدخل دائرة مغلقة إلى مالانهاية والحل الصحيح لهذه المشكلة أن لغة جافا تستخدم الكلمة المحجوزة super لتحديد استدعاء الطريقة من الفصيلة العليا فتكون الصيغة الصحيحة لتنفيذ دالة الإيداع في الفصيلة الفرعية هي:

```
// override the BankAccount deposit method
        public void deposit(double amount)

                transactionCount ++;
                // now add amount to balance
                super.deposit(amount);
```

وبذلك يمكن تنفيذ دالة السحب بنفس الطريقة فتكون الصيغة الصحيحة لها هي

```
// override the BankAccount withdraw method
        public void withdraw(double amount)
```

```
            transactionCount ++;
        // now subtract amount from balance
        super.withdraw(amount);
```

تنفيذ الطريقة الجديدة deductFees

تستدعى هذه الطريقةنهاية كل شهر لتنفيذ جملة شرطية لإختبار إذا كان عدد عمليات الكائن أكبر من عدد العمليات المجانية فإذا تحقق الشرط تقوم الطريقة بحساب الأجر fees المطلوب على العمليات الزائدة وخصم الأجرة من حساب الكائن باستخدام الطريقة withdraw(fees) وقبل نهاية الطريقة تقوم بإعطاء قيمة صفر لعداد العمليات لبدء عمليات شهر جديد والصيغة الصحيحة لهذه الطريقة هي

```
// New method deductFees
public void deductFees()

    if(transactionCount>FREE_TRANSACTIONS)

    double fees = TRANSACTION_FEE*(transactionCount- FREE_TRANSACTIONS);
    super.withdraw(fees);

    transactionCount = 0;
```

تنفيذ الحساب الثالث TimeDepositAccount

هذا النوع من الحساب هو امتداد للنوع الثاني SavingAccount لأنه يقوم بحساب العائد الشهري على الإيداع ولكن الاختلاف في هذا الحساب أن المودع يعد بترك المال لعدة شهور بدون سحب مقابل زيادة في معدل العائد ويتم خصم قيمة جزائية إذا تم سحب قبل انتهاء المدة ويبين الشكل (١ -٧) تنفيذ الفصيلة TimeDepositAccount وفيها يتم تعريف متغير كائن جديد periodsToMaturity لتحديد عدد شهور الإيداع ويتم إعطاء قيمة لهذا المتغير عند إنشاء كائن من هذه الفصيلة ويتم إنقاص هذا العداد في نهاية كل شهر عند حساب العائد في الطريقة addIntrest

من المخطط الهرمي شكل(١ -٥) يجب ملاحظة أن الفصيلة TimeDepositAccount تبعد مستويين عن الفصيلة BankAccount ولكن تظل الفصيلة BankAccount فصيلة عليا ولكنها ليست الفصيلة العليا المباشرة للفصيلة TimeDepositAccount ولكنها ترث منها الطريقتين getBalance و deposit

ولذلك يمكن القول أنه يمكن توريث الطرق من فصيلة عليا غير مباشرة بشرط أن لايكون تم استبدالها في الفصائل الفرعية بينهما.

تنفيذ طرق الفصيلة TimeDepositAccount

في هذه الفصيلة يوجد طريقتان

الطريقة الأولى addInterest() وهي طريقة مستبدلة override method للطريقة الموجودة في الفصيلة العليا SavingAccount لحساب العائد الشهري وإنقاص عدد شهور فترة الإيداع فيكون تنفيذ الطريقة كالتإلى

```
// override the SavingAccount addInterest method
        public void addInterest()

                periodsToMaturity--;
                super.addInterest();
```

الطريقة الثانية withdraw(double amount) وفيها تستخدم جملة شرطية لاختبار عدد شهور الإيداع إذا كان أكبر من الصفر تقوم الطريقة بخصم القيمة الجزائية ثم سحب القيمة المطلوبة وإن لم يتحقق الشرط تقوم بسحب القيمة فقط ولذلك يكون تنفيذ الطريقة كالتإلي:

```
// override the BankAccount withdraw method
        public void withdraw(double amount)

                 if (periodsToMaturity >0)
                         super.withdraw(EARLY_WITHDRAW_PENALITY);
                // now subtract amount from balance
                super.withdraw(amount);
```

TimeDepositAccount.java

```
/* The TimeDepositAccount extends SavingAccount class overrides
  the SavingAccount addInterest method and the BankAccount withdraw
  method to model an account like SavingAccount but you promize to leave
  the mony in the account for a particular number of months, and
  there is a penalty for early withdrawal.
*/
public class TimeDepositAccount extends SavingAccount
// The TimeDepositAccount constructor
{
          public TimeDepositAccount(double rate , int maturity)
          {
                    super (rate);
                    periodsToMaturity = maturity;
          }
// override the SavingAccount addInterest method
          public void addInterest()
          {
                    periodsToMaturity--;
                    super.addInterest();
          }
          // override the BankAccount withdraw method
          public void withdraw(double amount)
          {
                     if (periodsToMaturity >0)
                              super.withdraw(EARLY_WITHDRAW_PENALITY);
                    // now subtract amount from balance
                    super.withdraw(amount);
          }
// The TimeDepositAccount instance variables
          private int periodsToMaturity;
          private static final double EARLY_WITHDRAW_PENALITY = 20.0;
}
```

شكل (١ -٧)

أثناء تنفيذ طرق الفصيلة TimeDepositAccount لاحظ أن هذه الطرق تستخدم الجملة super.addIntrest(); و الجملة super.withdraw(amount); لإستدعاء الطرق من الفصيلة العليا. بالنسبة للطريقة addIntreset تستدعي من الفصيلة العليا المباشرة SavingAccount ولكن بالنسبة للطريقة withdraw فإن الفصيلة العليا المباشرة لاتملك الطريقة withdraw ولذلك تورث هذه الطريقة وتستبدل من الفصيلة العليا غير المباشرة BankAccount وبصفة عامة تستدعى الطرق من الفصيلة الأقرب في المخطط الهرمي للوراثة.

منشآت الفصيلة الفرعية Subclass constructors

عند إنشاء كائن من الفصيلة CheckingAccount يجب إعطاؤه قيمة للمتغير balance ولذلك يجب استدعاء منشىء الفصيلة العليا BankAccount لإعطاء قيمة للمتغير balance ولذلك يجب استخدام الكلمة المحجوزة super ويتبعها معاملات المنشىء بين أقواس ويجب أن تكون هذه الجملة هي الجملة الأولى في منشىء الفصيلة الفرعية ويكون تنفيذ منشىء الفصيلة الفرعية كالتالي:

```
// The CheckingAccount constructor

        public CheckingAccount(double initialBalance)

                // construct superclass
                super(initialBalance);
                // initialize transaction count
                 transactionCount = 0;
```

إذا لم يستدع منشىء الفصيلة الفرعية منشىء الفصيلة العامة فإن المنشىء الإفتراضى للفصيلة العامة يستدعى تلقائياً وإن لم يكن هناك منشىء افتراضي في الفصيلة العامة فإن المترجم يعطي خطأ ويمكن تنفيذ منشىء الفصيلة الفرعية CheckingAccount بدون استدعاء منشىء الفصيلة العليا BankAccount وحينئذ يستدعى المنشىء الإفتراضى للفصيلة BankAccount ويعطى قيمة صفر للمتغير balance وبعد ذلك يقوم منشىء الفصيلة الفرعية CheckingAccount بإيداع القيمة الأولية للمتغير balance ويكون تنفيذ المنشىء كالتإلي:

```
// The CheckingAccount constructor

        public CheckingAccount(double initialBalance)

        // superclass has been constructed with its default //constructor
        // now set initial balance
                super.deposit(initialBalance);
                // initialize transaction count
                 transactionCount = 0;
```

ولكن في حالة الفصيلة TimeDepositAccount ليس لنا اختيار حيث إن الفصيلة العليا SavingAccount ليس لها منشىء افتراضي ولذلك يجب استدعاء منشىء الفصيلة العليا كالتإلي

```
// The TimeDepositAccount constructor

        public TimeDepositAccount(double rate , int maturity)

                super (rate);
                periodsToMaturity = maturity;
```

تعدد الأشكال Polymorphism

علاقة الوراثة أحياناً تسمى "is – a" كل كائن من الفصيلة الفرعية يكون أيضاً كائناً من الفصيلة العليا ولكن بخصائص خاصة وهذا يعنى أن كل كائن من الفصيلة ChechingAccount هو كائن من الفصيلة BankAccount ولذلك يمكن استخدام كائن من الفصيلة الفرعية مكان كائن من الفصيلة العليا

مثال ذلك الطريقة transfer في فصيلة BankAccount تستخدم لتحويل مبلغ من المال من حساب إلى حساب آخر وتنفيذ هذه الطريقة كالتإلي

```
// The transfer method withdraw an amount from this object and //deposit to other object balance
        public void transfer(BankAccount other , double amount)

                withdraw(amount);
                other.deposit(amount);
```

وحيث إن جميع فصائل البنك هي امتداد للفصيلة BankAccount يمكن تمرير أي كائن من أي حساب للطريقة transfer

```
Ex:
    BankAccount c1 = new BankAccount(1000);
    CheckingAcount c2 = new CheckingAccount(2000);
    C1.transfer(c2,500);
```

دعنا نتتبع استدعاء هذه الطريقة بدقة. داخل استدعاء هذه الطريقة يوجد متغيرين من نوع BankAccount ولكنهما يشيرا إلى كائن BankAccount وكائن CheckingAccount كما في شكل(١ -٨) ولنأخذ في اعتبارنا استدعاء الطريقة deposit أي طريقة deposit المعامل other من نوع فصيلة Bankaccount ولذلك فإن الظاهر هو استدعاء الطريقة BankAccount.deposit ومن ناحية آخرى نجد أن الفصيلة CheckingAccount تملك الطريقة deposit التي تحدث متغير أعداد العمليات transactionCount وحيث إن المتغير other في الطريقة transfer فعليا يشير كائن الفصيلة الفرعية CheckingAccount يكون من المناسب لو أن الطريقة CheckingAccount.depoit تستدعى بدلاً من الطريقة BankAccount.deposit

وفي لغة جافا استدعاء الطريقة دائماً يحدد بنوع المتغير الفعلي actual object وليس نوع المرجعية للمتغير reference object وهذا يعني أن نفس الجملة other.deposit يمكن أن تنادي طرق مختلفة وهذا المبدأ أو الأساس يسمى تعدد الأشكال وهو المبدأ الثالث من مبادىء برمجة الكائنات OOP. والتعبير polymorphism يأتى من الكلمات الإغريقية many shapes

في لغة الجافا جميع طرق الكائنات instance methods تكون متعددة الأشكال polymorphic

في لغة الجافا أيضاً يمكن أن يكون هناك عدة طرق في فصيلة واحدة لها نفس الاسم ولكن المعاملات الظاهرة explicit parameters للطرق مختلفة وهذا ما يسمى بطرق التحميل الذائد overloaded methods ومثال ذلك المنشآت دائماً لها نفس الاسم ولكن تختلف المعاملات وحينئذ يختار المترجم compiler الطريقة المناسبة عند ترجمة البرنامج طبقاً لتطابق المعاملات الظاهرة explicit parameters للطريقة المستدعاة
هناك فرق بين تعدد الأشكال polymorphism والتحميل الزائد overload وهو أن في حالة التحميل الزائد overload المترجم يحدد الطريقة أثناء ترجمة البرنامج أي قبل تنفيذ البرنامج وهذا الاختيار للطريقة يسمى رابط مبكر early binding ولكن في حالة اختيار الطريقة التي تطابق نوع المعامل الضمني كما في حالة الطريقة deposit التي تم تحليلها سابقاً فإن المترجم compiler لايأخذ أي قرار عند ترجمة البرنامج ويتم تنفيذ البرنامج قبل أن يعرف ماذا يخزن في المتغير other ولذلك فإن الآلة التخيلية virtual machine وليس المترجم compiler هي التي تختار الدالة المناسبة وهذا الاختيار يسمى الرابط المتآخر late binding

البرنامج BankAccountTest في شكل (١ -٨) و البرنامج polymorphismTest في شكل(١ -٩) هي برامج تطبيقية يجب على القارىء تتبع تنفيذها يدوياً قبل استعراض نتائجها من الحاسب لتأكيد المفاهيم التي تم شرحها في هذه الوحدة.

BankAcountTest.java

```
public class BankAcountTest
{
        /* This program is used to test the diferent bankAcounts          classes
        */
        // The main method
        public static void main(String [] args)
        {
                BankAccount mohamed = new BankAccount();
                BankAccount ahmed = new BankAccount(1000);
                BankAccount mahmoud = new BankAccount(2000);
                mohamed.deposit(3000);
                ahmed.deposit(3000);
                mahmoud.deposit(3000);
                // transfer 1000Sr from mahmoud to ahmed
                mahmoud.transfer(ahmed,1000);
                System.out.println(mohamed.getBalance());
                System.out.println(ahmed.getBalance());
                System.out.println(mahmoud.getBalance());
                // construct new SavingAccount object called moustafa
                SavingAccount moustafa = new SavingAccount(5);
                moustafa.deposit(1000);
                moustafa.addInterest();
                System.out.println(moustafa.getBalance());
        // construct new CheckingAccount object called baker
                CheckingAccount baker = new CheckingAccount(5000);
                baker.deposit(1000);
                baker.deposit(1000);
                baker.deposit(1000);
                baker.deposit(1000);
                baker.deposit(1000);
                baker.withdraw(2000);
                baker.deductFees();
                System.out.println(baker.getBalance());
        // construct new TimeDepositAccount object called flah
                TimeDepositAccount flah = new   TimeDepositAccount(5,5);
                flah.deposit(10000);
                System.out.println(flah.getBalance());
                flah.addInterest();
                System.out.println(flah.getBalance());
                flah.addInterest();
                System.out.println(flah.getBalance());
                flah.deposit(1000);
                System.out.println(flah.getBalance());
                flah.addInterest();
                System.out.println(flah.getBalance());
                flah.withdraw(5000);
                System.out.println(flah.getBalance());
```

```
                flah.addInterest();
                System.out.println(flah.getBalance());
                flah.addInterest();
                System.out.println(flah.getBalance());
                flah.withdraw(5000);
                System.out.println(flah.getBalance());
        }
}
```

شكل (١ -٨)

PolymorphismTest.java

```
public class PolymorphismTest
{
        /* This program is used to test the principles of polymorphism

        */


        // The main method

        public static void main(String [] args)
        {
                // construct new SavingAccount object called abdalah
                SavingAccount abdalah = new SavingAccount(0.5);

                // construct new TimeDepositAccount object called flah
                TimeDepositAccount flah = new TimeDepositAccount(1,3);

                // construct new CheckingAccount object called baker
                CheckingAccount baker = new CheckingAccount(0);

                abdalah.deposit(10000);

                flah.deposit(10000);

                printBalance("abdalah ",abdalah);
                printBalance("flah    ",flah);
                printBalance("baker   ",baker);

                // transfer 1000Sr from abdalah to baker

                abdalah.transfer(baker,2000);

                // transfer 1000Sr from flah to baker

                flah.transfer(baker,980);
```

```
                printBalance("abdalah ",abdalah);
                printBalance("flah    ",flah);
                printBalance("baker   ",baker);


                baker.withdraw(500);
                printBalance("baker   ",baker);
                baker.withdraw(80);
                printBalance("baker   ",baker);
                baker.withdraw(400);
                printBalance("baker   ",baker);

                endOfMonth(abdalah);
                endOfMonth(flah);
                endOfMonth(baker);
                printBalance("abdalah ",abdalah);
                printBalance("flah    ",flah);
                printBalance("baker   ",baker);
        }
        public static void endOfMonth(SavingAccount savings)
        {savings.addInterest();
        }
        public static void endOfMonth(CheckingAccount checking)
        {checking.deductFees();
        }
        public static void printBalance(String name , BankAccount account)
        {System.out.println("The balance of " + name +
         "account is " + account.getBalance()+"  SR");
        }
}
```

شكل (١ -٩)

نتائج تنفيذ البرنامج PolymorphismTest

```
The balance of abdalah account is 10000.0  SR
The balance of flah    account is 10000.0  SR
The balance of baker   account is 0.0  SR
The balance of abdalah account is 8000.0  SR
The balance of flah    account is 9000.0  SR
The balance of baker   account is 2980.0  SR
The balance of baker   account is 2480.0  SR
The balance of baker   account is 2400.0  SR
The balance of baker   account is 2000.0  SR
The balance of abdalah account is 8040.0  SR
The balance of flah    account is 9090.0  SR
The balance of baker   account is 1996.0  SR
```



برمجة ٣

معالجة الإستثناءات

معالجة الاستثناءات

٢

الجدارة:

أن يكون المتدرب قادراً على فهم أساسيات وأنواع الاستثناءات في لغة الجافا والقدرة على كتابة برامج تحتوي على معالجة هذه الاستثناءات

الأهــداف:

١. فهم أساسيات معالجة الاستثناءات في لغة الجافا
٢. تعلم أنواع الاستثناءات
٣. استخدام التعليمة try لاحتواء جزء البرنامج الذي يمكن أن يحدث به استثناء
٤. استخدام التعليمة catch لمعالجة الأنواع المختلفة للاستثناءات
٥. استخدام التعليمة finally
٦. القدرة على تمرير استثناء

مستوى الأداء المطلوب:

أن يصل المتدرب إلى اتقان الجدارة بنسبة ١٠٠٪

الوسائل المساعدة:

- وجود حاسب آلي
- وجود بيئة متكاملة لبناء وتنفيذ برامج لغة الجافا
- دفتر
- قلم

معالجة الاستثناءات

Exception Handling

مقدمة :

الاستثناء هو مؤشر لحدوث خطأ أثناء عملية تنفيذ البرنامج مما يؤدي إلى تعطيل التسلسل الطبيعي لتعليمات البرنامج وقد تعلمنا في الفصل السابق أن الوراثة في لغة الجافا تعطيها صفة الامتدادية وهذه الصفة يمكن أن تزيد من عدد ونوع الأخطاء التي يمكن أن تحدث حيث إن كل فصيلة جديدة تضاف إلى البرنامج يمكن أن تضيف مصدراً من مصادر الاستثناءات في البرنامج. إذاً نستطيع القول أن الاستثناء هو حدوث خطأ ما وهذا الخطأ ليس خطأ في بناء الجملة syntax error ولكنه قد يكون له العديد من المصادر مثل القسمة على صفر ومعاملات غير متاحة للدالة و الإشارة إلى عنصر في المصفوفة خارج نطاقها.

عند حدوث استثناء يحتاج البرنامج إلى معالجة هذا الاستثناء لكي يستمر تنفيذ البرنامج بصورة طبيعية وسابقا قبل عام ١٩٩٠ كانت معالجة الاستثناءات تتم باختبار قيم صحيحة تعود بدلائل مثل القيمة صفر تدل على النجاح والقيمة السالبة تدل على نوع من الاستثناءات وهذه القيم أصبحت تعرف بشفرات الأخطاء Error codes وقد تم اكتشاف أن استخدام هذا النوع من معالجة الأخطاء يتسبب في ثلاث مشاكل:

١ - غالباً تهمل شفرة الخطأ

٢ - اختبار شفرة الأخطاء تعترض التدفق الطبيعي للبرنامج مما يصعب تتبع المستخدم للبرنامج

٣- اختبار شفرة الأخطاء يزيد من حجم البرنامج

أساسيات معالجة الاستثناء في لغة الجافا

The basics of java Exception Handling

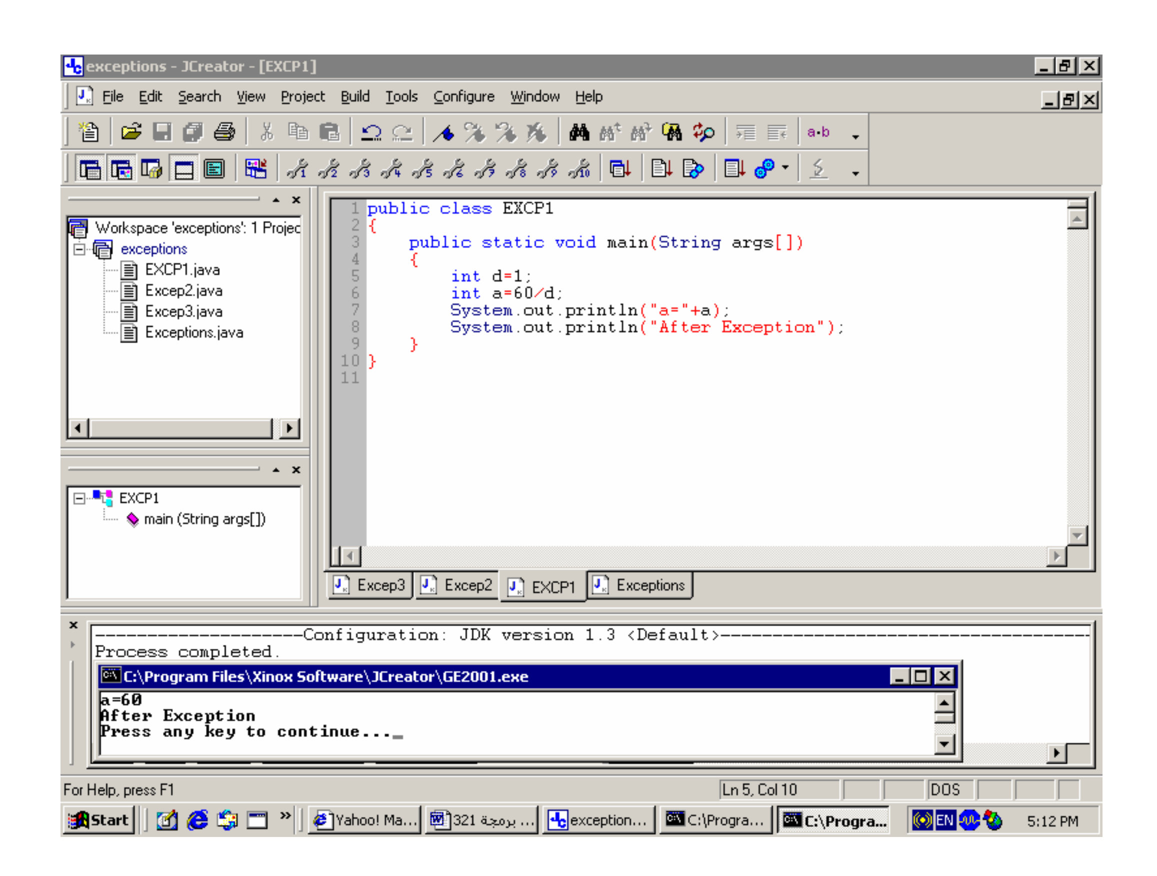
لقد أدت مشاكل استخدام شفرة الأخطاء Error codes إلى تطوير آلية جديدة لمعالجة الاستثناءات في لغة الجافا تعتمد على الكائنات مما أدى إلى برامج سهلة القراءة والتتبع وكذلك برامج أكثر مرونة.

وفي هذا النموذج عند حدوث استثناء أثناء تشغيل برنامج الجافا إما البرنامج program أو آلة لغة الجافا الافتراضية JVM تنشىء كائن لوصف الاستثناء ويشمل هذا الكائن قيم المتغيرات في لحظة حدوث الاستثناء.

إذا تم إنشاء الكائن من البرنامج فإن البرنامج يمرر ذلك الكائن إلى آلة الجافا الافتراضية JVM وعند استقبال الكائن تبحث في البرنامج عن معالج الاستثناء handler exception الذى يمكن أن يعالج الاستثناء الموصوف بالكائن. إذا وجد المعالج يتم تمرير الكائن لمعالج الاستثناء الذي يقوم باستخدام محتويات الكائن لمعالجة الاستثناء. إذا لم يوجد معالج الاستثناء يتوقف البرنامج عن التنفيذ.

مثال ١

شكل (٢ -١) يبين برنامج بسيط لقسمة رقمين إذا تتبعنا هذا البرنامج نجد أنه يتم تنفيذه بطريقة صحيحة لأنه يوجد قيمة غير صفرية للمقام (d)

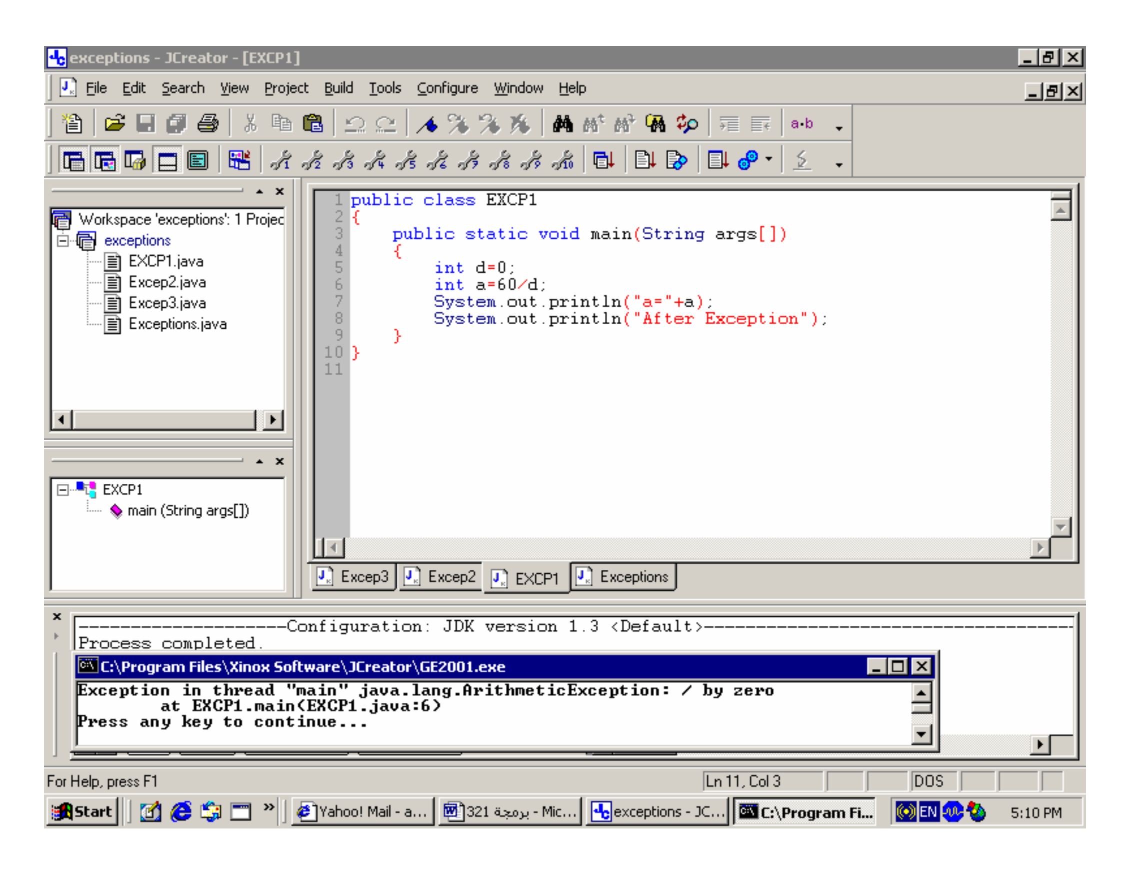


شكل (٢-١)

ولكن شكل (٢ -٢) يبين نفس البرنامج ولكن تم إعطاء المقام قيمة صفرية ولذلك اذا تتبعنا الشكل نجد أن المترجم (compiler) قد أنهي ترجمة البرنامج بنجاح وهذا ملاحظ من الجملة Process completed ولكن عند تنفيذ البرنامج هناك خطأ تنفيذي (استثناء) وهو القسمة على صفر فقد بحث عن معالج للخطأ في البرنامج فلم يجد ولذلك أعطى رسالة

Exception in thread "main" java.lang.ArithmeticException: /by zero at Excp1.main <Excp1.java:6>

وهذا يبين أن كائن الاستثناء من الفصيلة ArithmaticException ويشمل بيان الخطأ وهو القسمة على صفر by zero/ وقد بحث عن معالج داخل البرنامج فلم يجد وتلاحظ في شكل (٢ -٢) إنهاء تنفيذ البرنامج قبل تنفيذ جمل الطباعة



شكل (٢-٢)

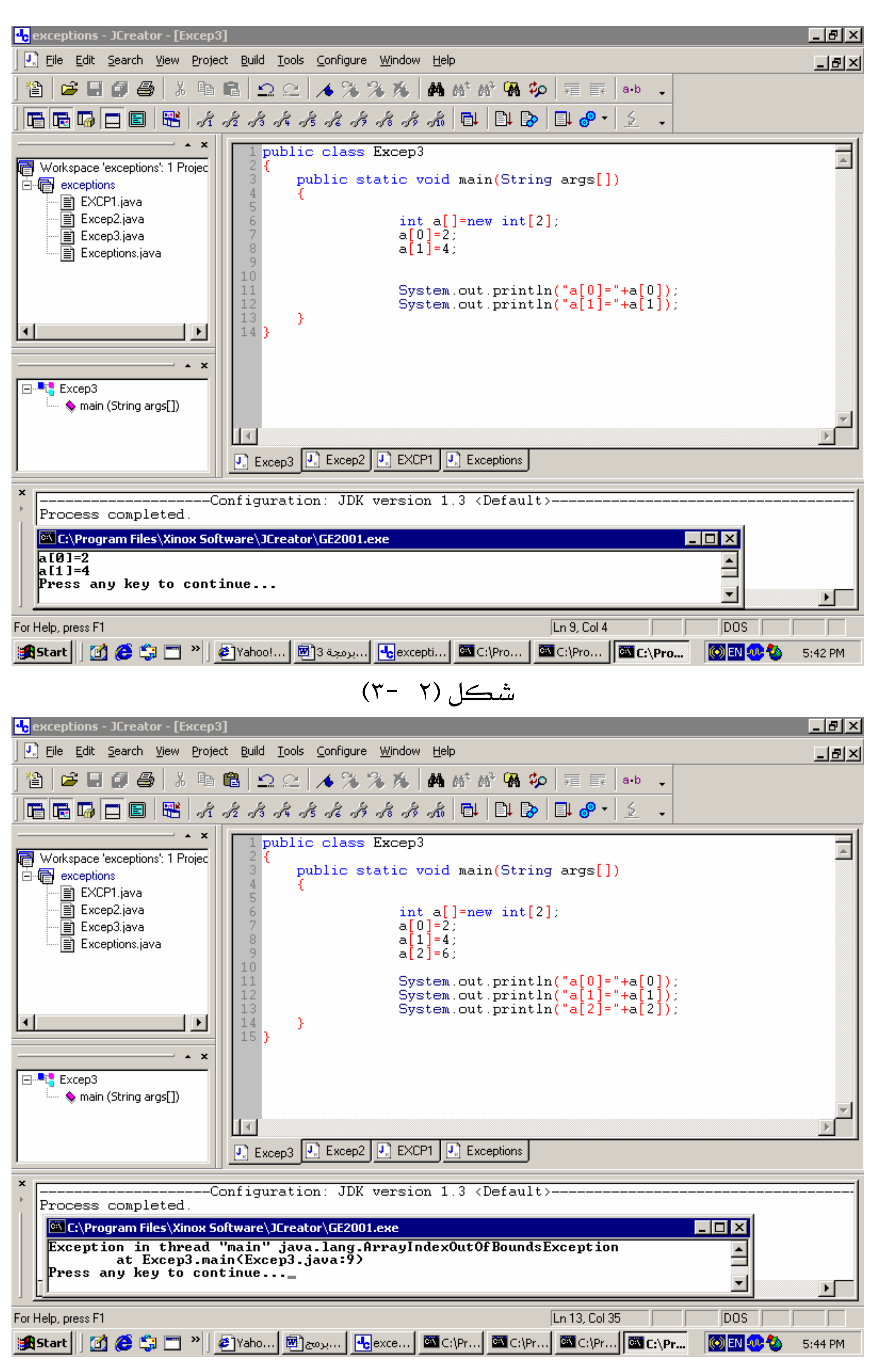
مثال ٢

يبين شكل (٢ -٣) برنامج بسيط لتعريف مصفوفة مكونة من عنصرين من نوع الأرقام الصحيحة وإعطاء قيم للعنصرين وطباعة قيم العنصرين تلاحظ من الشكل تنفيذ البرنامج بصورة طبيعية وتم إعطاء نتائج الطباعة.

ولكن شكل (٢ -٤) يبين نفس البرنامج مع إضافة الجملة a[2] = 6; ونتيجة أن هذا العنصر خارج نطاق تعريف المصفوفة فقد وقع استثناء وتم البحث عن معالج داخل البرنامج فلم يوجد فتوقف تنفيذ البرنامج قبل تنفيذ جمل الطباعة وأعطى رسالة الخطأ التالية:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundException at Excp1.main <Excp3.java:9>

وهذا يبين أن كائن الاستثناء من الفصيلة ArrayIndexOutOfBoundException


exceptions - JCreator - [Excep3]
File Edit Search View Project Build Tools Configure Window Help
Workspace 'exceptions': 1 Projec
exceptions
EXCP1.java
Excep2.java
Excep3.java
Exceptions.java
Excep3
main (String args[])
public class Excep3
{
public static void main(String args[])
{
int a[]=new int[2];
a[0]=2;
a[1]=4;
System.out.println("a[0]="+a[0]);
System.out.println("a[1]="+a[1]);
}
}
Excep3 Excep2 EXCP1 Exceptions
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
a[0]=2
a[1]=4
Press any key to continue...
For Help, press F1
Ln 9, Col 4
DOS
Start
5:42 PM

شكل (٢ -٣)

exceptions - JCreator - [Excep3]
File Edit Search View Project Build Tools Configure Window Help
Workspace 'exceptions': 1 Projec
exceptions
EXCP1.java
Excep2.java
Excep3.java
Exceptions.java
Excep3
main (String args[])
public class Excep3
{
public static void main(String args[])
{
int a[]=new int[2];
a[0]=2;
a[1]=4;
a[2]=6;
System.out.println("a[0]="+a[0]);
System.out.println("a[1]="+a[1]);
System.out.println("a[2]="+a[2]);
}
}
Excep3 Excep2 EXCP1 Exceptions
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
at Excep3.main(Excep3.java:9)
Press any key to continue...
For Help, press F1
Ln 13, Col 35
DOS
Start
5:44 PM

شكل (٢ -٤)

أنواع الاستثناءات Exception types

من الأمثلة السابقة تبين أن هناك العديد من أنواع الاستثناءات ومن معرفتنا للغة الجافا بأنها تتكون من فصائل فإن الاستثناءات في الجافا هي فصائل classes وكل فصيلة class تختص بنوع من الاستثناءات وجميع هذه الفصائل ترث الفصيلة العليا Throwable وتوجد فصيلتان فرعيتان ترثان هذه الفصيلة وهما Exception subclass و Error subclass وهذه الفصائل موجودة في الحزمة java.lang وهذه الفصائل الفرعية تصنف الاستثناءات أهي ذات علاقة بالبرنامج program related أم هي ذات علاقة بآلة الجافا الافتراضية JVM

الفصيلة Exception هي الفصيلة الجزرية root class لجميع الفصائل التي تصف جميع الاستثناءات ذات العلاقة ببرنامج جافا ويبين الجدول (٢ -١) بعض الفصائل الفرعية للفصيلة Exception ووصف كل منها.

الجدول (٢ -١) بعض الفصائل الفرعية للفصيلة Exception

الكائن المنشىء من هذه الفصيلة يصف مشكلة حسابية مثل محأولة القسمة على صفر	ArithmaticException
الكائن المنشىء من هذه الفصيلة يصف محأولة عنصر في مصفوفة بدليل غير معرف	ArrayIndexOutOfBoundsException
الكائن المنشىء من هذه الفصيلة يصف محأولة تحميل ملف فصيلة غير موجودة	ClassNotFoundException
الكائن المنشىء من هذه الفصيلة يصف محأولة ملف غير موجود	FileNotFoundException
الكائن المنشىء من هذه الفصيلة يصف خطأ عاماً أثناء عملية إدخال أو آخراج وتوجد فصائل فرعية من هذه الفصيلة لتصف الخطأ بدقة	IOException
الكائن المنشىء من هذه الفصيلة يصف محأولة استدعاء طريقة متغير كائن ليس له مرجعية لأي كائن null reference	NullPointerException

الفصيلة Error هي الفصيلة الجزرية root class لجميع الفصائل التي تصف الاستثناءات ذات العلاقة بآلة الجافا الافتراضية JVM مثال ذلك الفصيلة OutOfMemoryError class فإن الكائن المنشىء من هذه الفصيلة يصف محأولة خاطئة لتخصيص ذاكرة.

معالجة الاستثناءات في الجافا Exception Handling In java

تتم معالجة الأخطاء في لغة الجافا باستخدام مجموعة من التعليمات وهي:

a- try block
b- catch blocks
c- finally block
d- throw statement
e- throws clause

أولاً : try…catch finally blocks

تستخدم لغة الجافا التعليمة try لتحديد الجزء من البرنامج الذي يحتمل أن يحدث به خطأ ويجب أن يتبع هذا الجزء مباشرةً تعليمة catch أو أكثر لتحديد طريقة معالجة الأنواع المتوقعة من الاستثناءات ثم يتبع آخر تعليمة catch تعليمة finally وهي اختيارية وتستخدم لتحديد جزء من البرنامج يجب تنفيذه بغض النظر هل حدث استثناء في جزء تعليمة try أم لم يحدث استثناء ويكون الشكل العام للتعامل مع الاستثناءات في لغة الجافا كالتالي:

```
try

        // Tested statements

catch ( ExceptionType 1   exob1)

        // exception handler for ExceptionType1

catch ( ExceptionType 2   exob2)

        // exception handler for ExceptionType2

finally

        // Statements that must be executed
```

مثال ٣

شكل (٢ -٥) يبين نفس البرنامج قسمة رقمين الذى تم شرحه في المثال ١ وفي هذا الشكل يتم معالجة الاستثناء ArithmaticException الذي حدث عندما أعطى المقام d قيمة صفرية ويتم ذلك باستخدام التعليمة try لإحتواء الجمل التي تسببت في الاستثناء كالتإلي:

```
try

        int d=0;
        int a=60/d;
```

ويتبع ذلك تعليمة catch التي تعالج هذا النوع من الاستثناء كالتالي:

```
catch (ArithmeticException   e)

            System.out.println("divide by zero");
            System.out.println(e.getMessage());
```

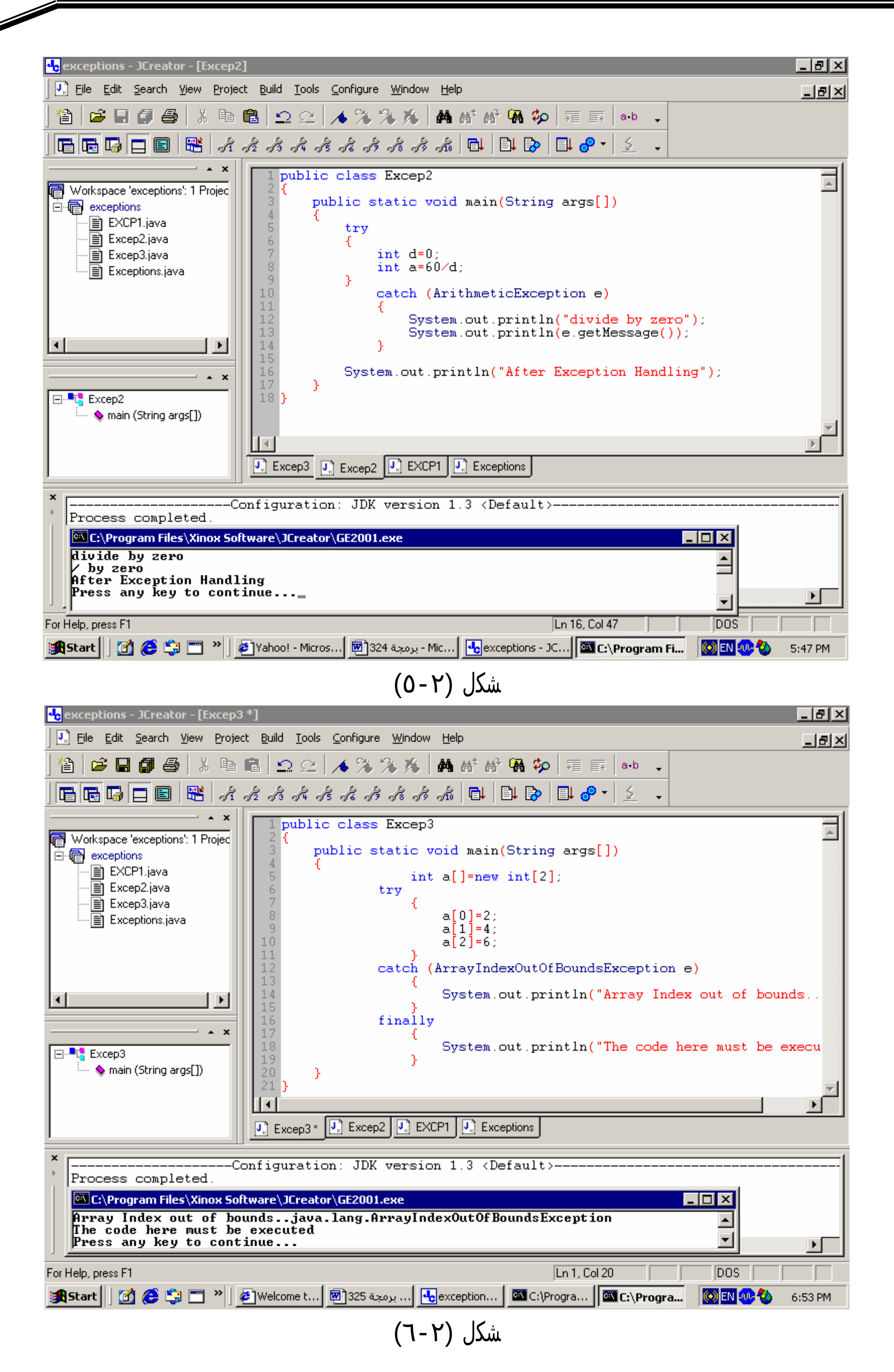
ويمكن تتبع تنفيذ البرنامج حيث تم حدوث استثناء عند تنفيذ السطر رقم ٨ في شكل (٢ -٥) وهو محأولة القسمة على صفر فتم إنشاء كائن من الفصيلة ArithmeticException وتم البحث عن معالج لهذا الاستثناء في البرنامج فوجد معالج مطابق للإستثناء في سطر ١٠ فتم انتقال تسلسل تنفيذ البرنامج إلى معالج الاستثناء الذي يحتوي على جملتين الأولى في السطر ١٣ لإظهار الرسالة divide by zero والثانية في السطر ١٤ لإظهار الرسالة التي يحتويها كائن الاستثناء e وهي by zero/ وبعد تنفيذ المعالج تم تنفيذ الجملة التي تتبع المعالج مباشرةً في سطر ١٦ لإظهار الرسالة After Exception Handling وتم إنهاء البرنامج بصفة طبيعية.

مثال ٤

شكل (٢ -٦) يبين نفس برنامج المصفوفة الذي تم شرحه في المثال ٢ وفي هذا الشكل يتم معالجة الاستثناء ArrayIndexOutOfBoundsException الذي حدث عند محأولة إعطاء عنصر المصفوفة a[2] قيمة ٦ في سطر ١٢ ويتم التقاط الاستثناء باستخدام التعليمة try لاحتواء الجمل التي تسببت في الاستثناء كالتالي:

```
try

            a[0]=2;
            a[1]=4;
            a[2]=6;
```



exceptions - JCreator - [Excep2]
public class Excep2
{
public static void main(String args[])
{
try
{
int d=0;
int a=60/d;
}
catch (ArithmeticException e)
{
System.out.println("divide by zero");
System.out.println(e.getMessage());
}
System.out.println("After Exception Handling");
}
}
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
divide by zero
/ by zero
After Exception Handling
Press any key to continue...
exceptions - JCreator - [Excep3 *]
public class Excep3
{
public static void main(String args[])
{
int a[]=new int[2];
try
{
a[0]=2;
a[1]=4;
a[2]=6;
}
catch (ArrayIndexOutOfBoundsException e)
{
System.out.println("Array Index out of bounds..
}
finally
{
System.out.println("The code here must be execu
}
}
}
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Array Index out of bounds..java.lang.ArrayIndexOutOfBoundsException
The code here must be executed
Press any key to continue...

شكل (٢-٥)

شكل (٢-٦)

ولمعالجة الاستثناء ArrayIndexOutOfBoundsException نستخدم تعليمة catch كالتالي:

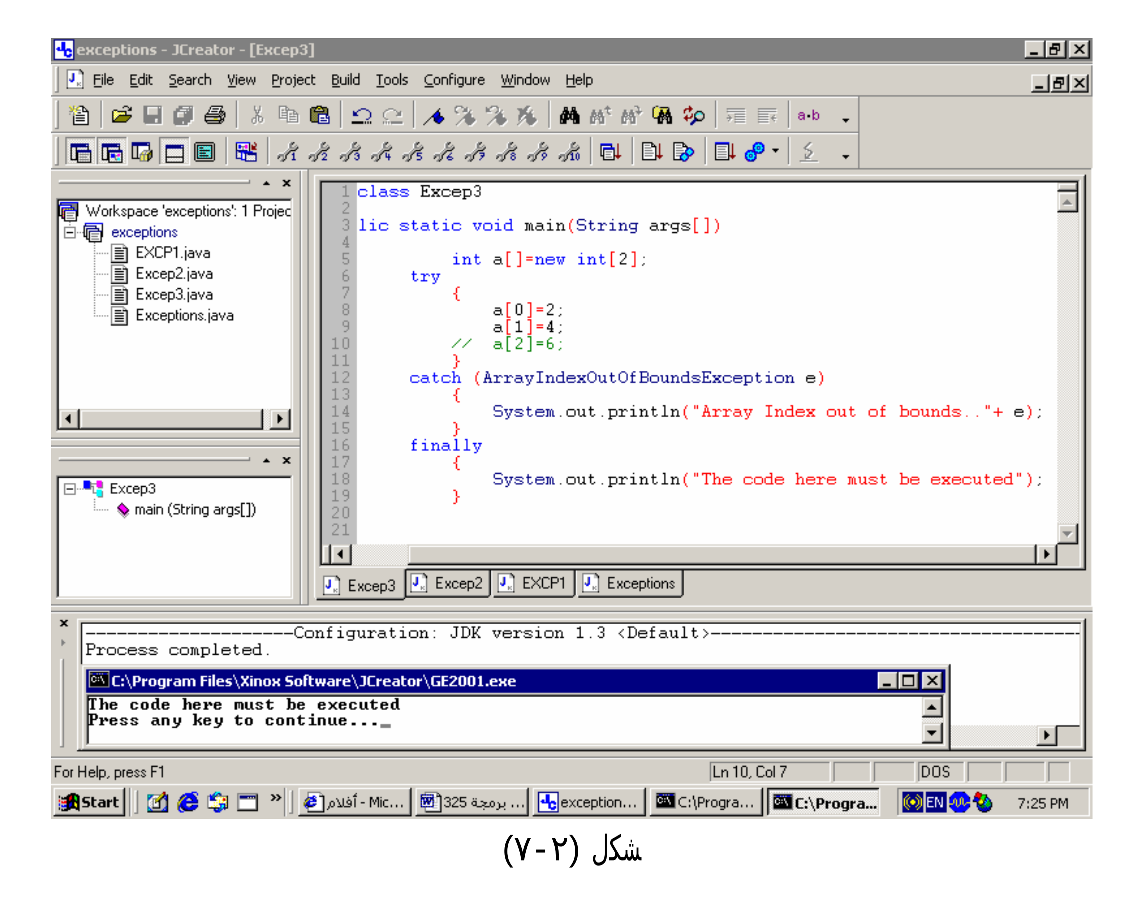
catch (ArrayIndexOutOfBoundsException e)

System.out.println("Array Index out of bounds.."+ e);

وفي هذا البرنامج تم استخدام تعليمة finally لإضاح كيف يمكن احتواء جزء البرنامج الذي يجب تنفيذه بعد تعليمة try سواء حدث استثناء أم لا.

وبتتبع تنفيذ هذا البرنامج في شكل (٢ -٦) نجد أنه تم تنفيذه بطريقة صحيحة حيث تم التقاط الاستثناء عند تنفيذ السطر ١٠ وتم البحث عن معالج الاستثناء وبدأ تنفيذه من السطر ١٢ وفيه تم إظهار الرسالة bounds of out Index Array بالإضافة إلى الذي يحتويها هو الكائن(e) وهو java.lang.ArrauIndexOutofBoundsException ثم بعد ذلك تم تنفيذ جزء البرنامج الذى تحتويه التعليمة finally وهو إظهار الرسالة :

The code here must be executed



شكل (٢-٧)

شكل (٢ -٧) يبين تنفيذ البرنامج السابق بعد إلغاء الجملة التي تسببت في الاستثناء فنجد أنه تم تنفيذ الجملة داخل التعليمة finally كما ذكرنا سابقا أنه يتم تنفيذ هذا الجزء من البرنامج سواء كان هناك استثناء أم لا.

تعدد تعليمة catch

عند توقع حدوث أكثر من استثناء في جزء من البرنامج فيمكن احتواء هذا الجزء بتعليمة try لالتقاط الأنواع المختلفة من الاستثناءات ثم نتبع ذلك بالعديد من تعليمات catch كلاً منها يعالج نوعاً من أنواع الاستثناءات المتوقعة.

مثال ٥

شكل (٢ -٨) يبين برنامج يجمع بين الجمل التي تستخدم لقسمة رقمين و الجمل التي تستخدم لتعريف مصفوفة وإعطاء قيم لعناصرها وفي هذا البرنامج تم إضافة التعليمة try لاحتواء هذه الجمل من البرنامج ثم اتبع ذلك بتعليمتين catch الأولى لمعالجة الاستثناء ArithmeticException والثانية لمعالجة الاستثناء ArrayIndexOutOfBoundsException .

ولتتبع تنفيذ هذا البرنامج نجد حدوث استثناء عند تنفيذ جملة القسمة في سطر ٨ وتم التقاط هذا النوع من الاستثناء والبحث له عن معالج فإنتقل تسلسل البرنامج إلى السطر ١٥ لتنفيذ المعالج وتم طباعة الرسائل

Handling the first Exception divide by zero
/ by zero

ثم انتقل تسلسل تنفيذ البرنامج إلى التعليمة finally وتم إظهار الرسالة

The code here must be executed

ملحوظة هامة: في المثال السابق يجب ملاحظة أن تسلسل تنفيذ البرنامج لم يعد مرة آخرى إلى جزء تعليمة try في السطر ٩ الذي يلي نقطة انتقال التسلسل للبحث عن معالج الاستثناء.

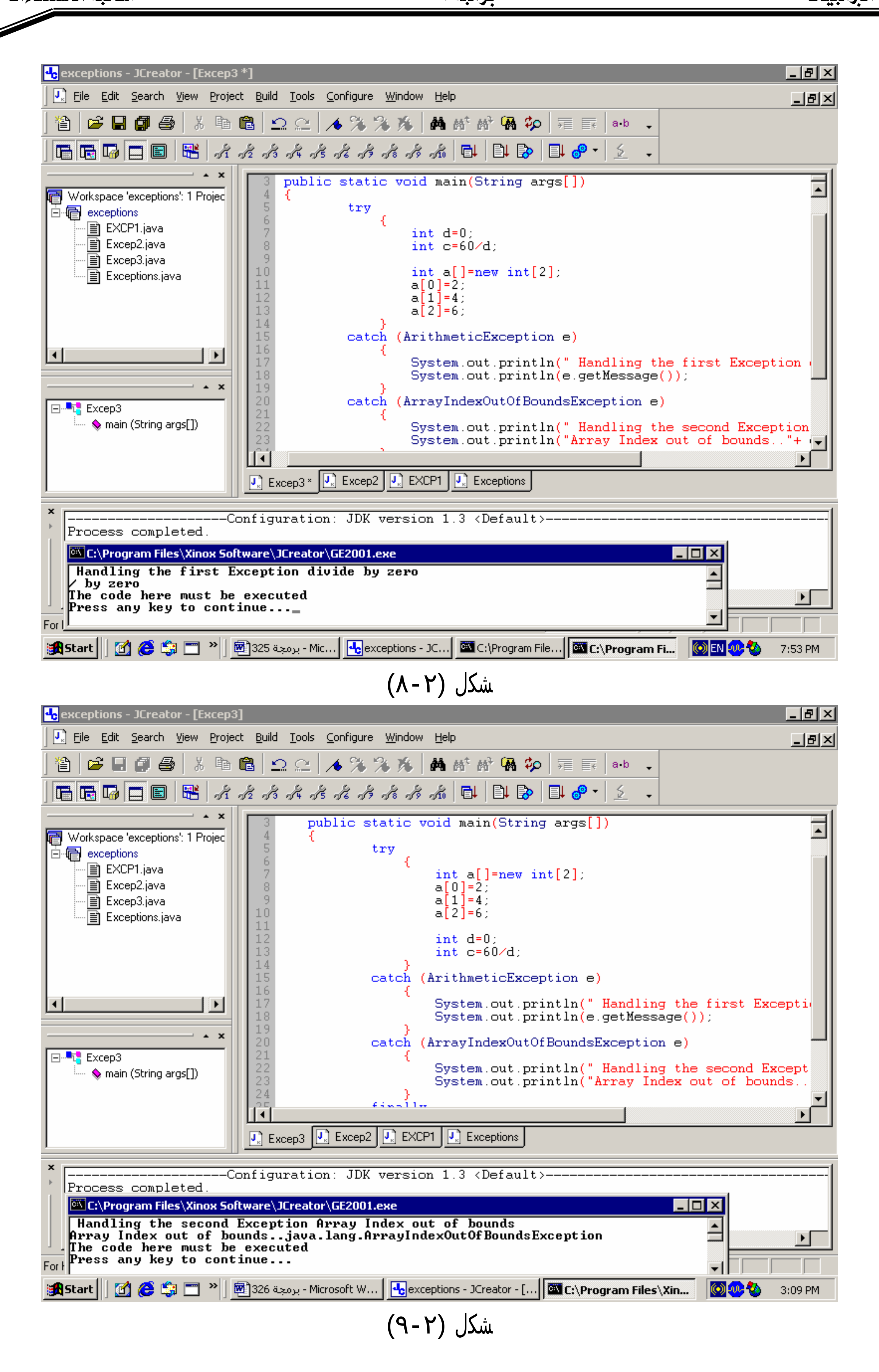
شكل (٢-٩) يبين نفس البرنامج مع تقدم جمل المصفوفة عن جمل قسمة رقمين ولذلك عند تنفيذ البرنـامج حـدث استثناء في السطر ١٠ وتم انتقال تسلسل البرنامج إلى السطر ٢٠ وإظهار الرسائل

Handling the second Exception Array Index out of bounds
Array Index out of bounds… java.lang.ArrauIndexOutofBoundsException

ثم تم الانتقال إلى تعليمة finally لإظهار الرسالة

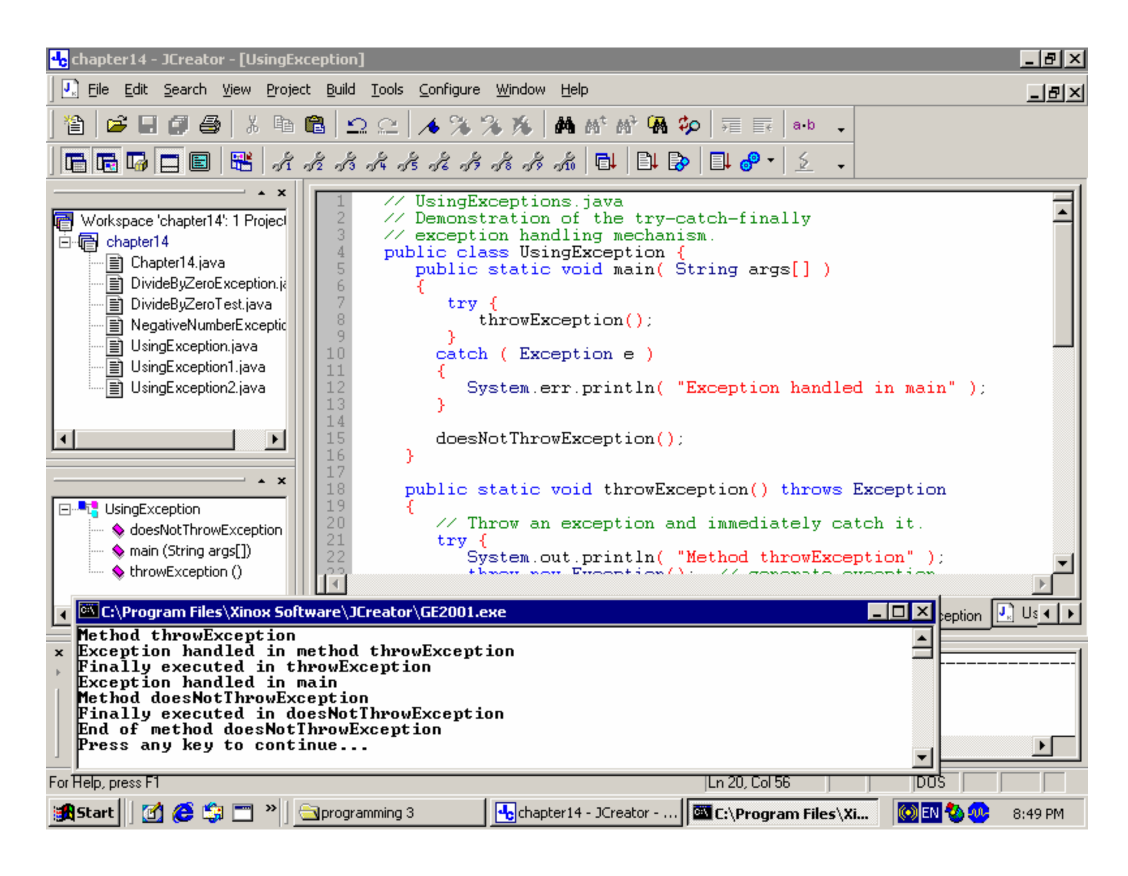
The code here must be executed

ولم يعد تسلسل البرنامج إلى تعليمة try مرة آخرى.

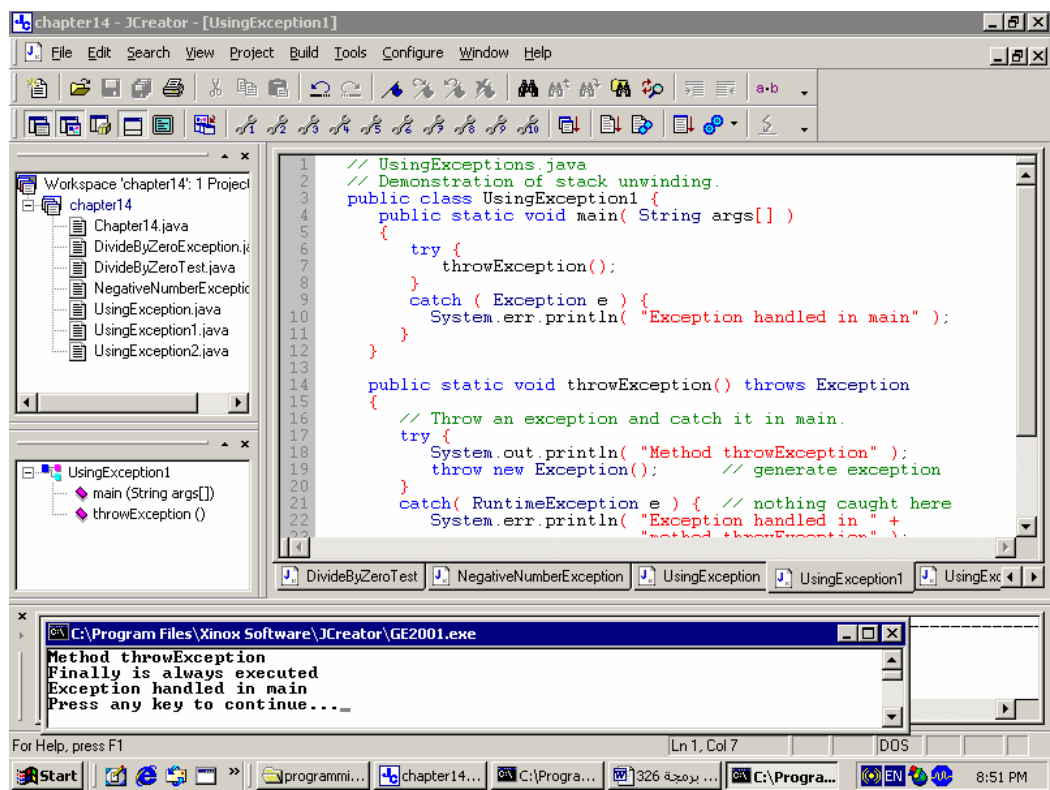

exceptions - JCreator - [Excep3 *]
File Edit Search View Project Build Tools Configure Window Help
Workspace 'exceptions': 1 Projec
exceptions
EXCP1.java
Excep2.java
Excep3.java
Exceptions.java
Excep3
main (String args[])
public static void main(String args[])
{
try
{
int d=0;
int c=60/d;
int a[]=new int[2];
a[0]=2;
a[1]=4;
a[2]=6;
}
catch (ArithmeticException e)
{
System.out.println(" Handling the first Exception
System.out.println(e.getMessage());
}
catch (ArrayIndexOutOfBoundsException e)
{
System.out.println(" Handling the second Exception
System.out.println("Array Index out of bounds.."+
Excep3 * Excep2 EXCP1 Exceptions
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Handling the first Exception divide by zero
/ by zero
The code here must be executed
Press any key to continue...
Start
7:53 PM
exceptions - JCreator - [Excep3]
File Edit Search View Project Build Tools Configure Window Help
public static void main(String args[])
{
try
{
int a[]=new int[2];
a[0]=2;
a[1]=4;
a[2]=6;
int d=0;
int c=60/d;
}
catch (ArithmeticException e)
{
System.out.println(" Handling the first Excepti
System.out.println(e.getMessage());
}
catch (ArrayIndexOutOfBoundsException e)
{
System.out.println(" Handling the second Except
System.out.println("Array Index out of bounds..
}
Excep3 Excep2 EXCP1 Exceptions
--------------------Configuration: JDK version 1.3 <Default>--------------------
Process completed.
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Handling the second Exception Array Index out of bounds
Array Index out of bounds..java.lang.ArrayIndexOutOfBoundsException
The code here must be executed
Press any key to continue...
Start
3:09 PM

شكل (٢-٨)

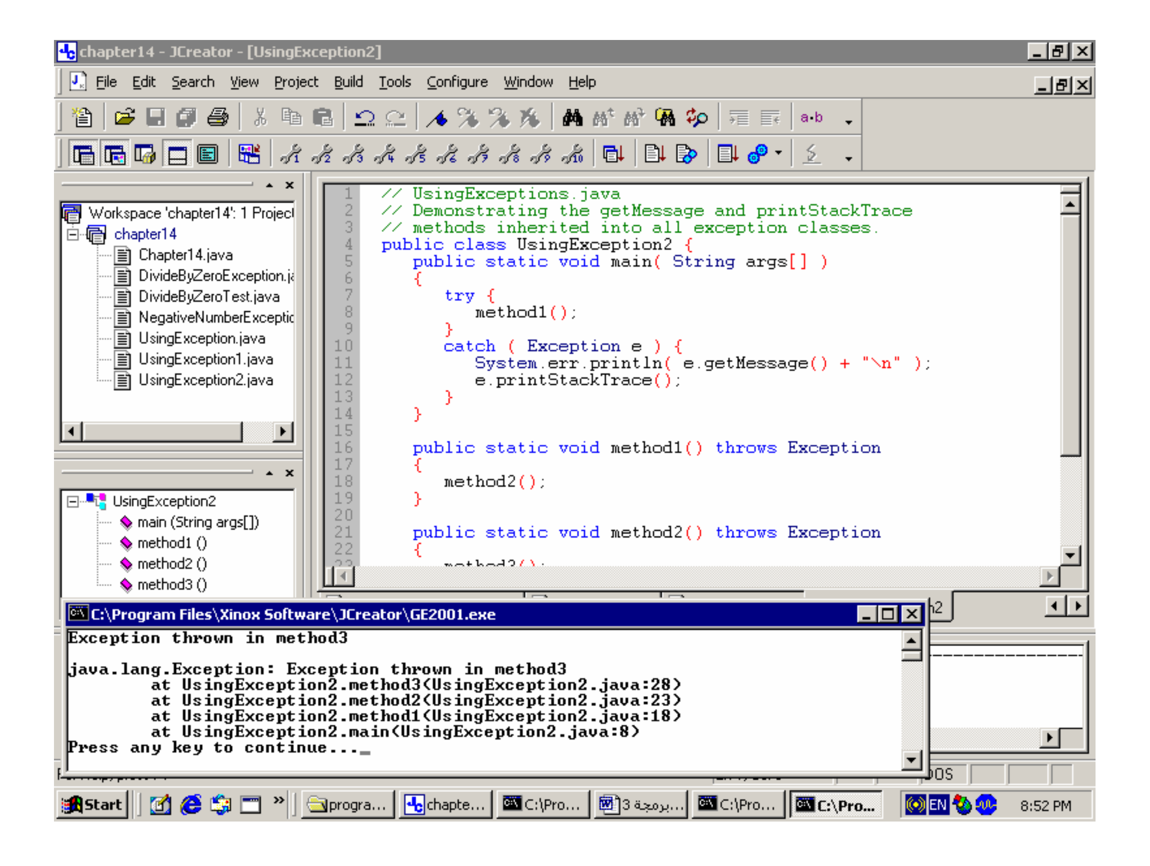
شكل (٢-٩)

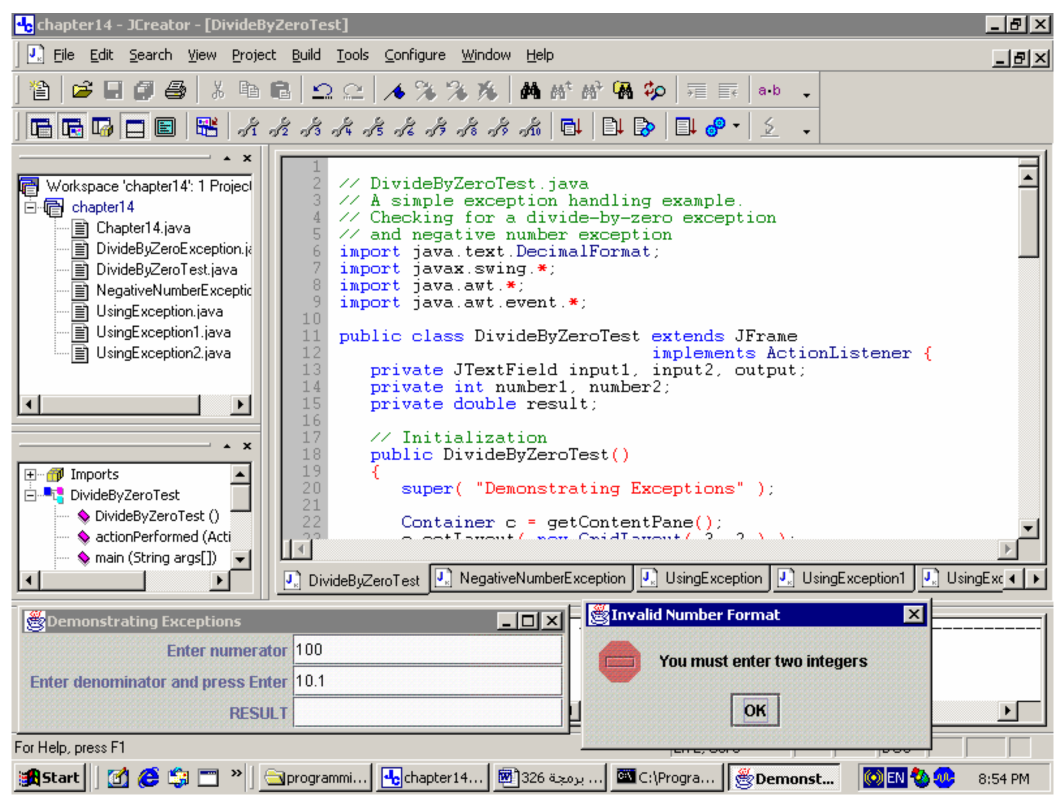


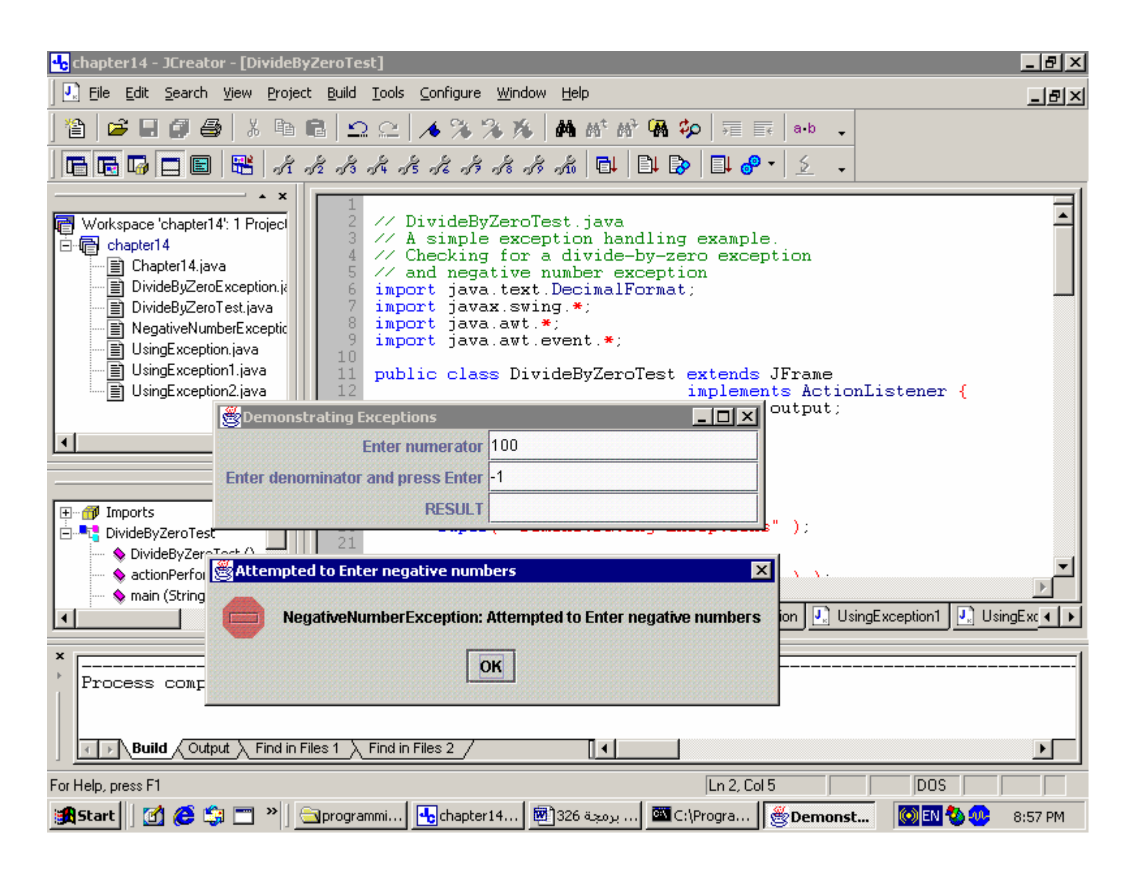
chapter14 - JCreator - [UsingException]
File Edit Search View Project Build Tools Configure Window Help
Workspace 'chapter14': 1 Project
chapter14
Chapter14.java
DivideByZeroTest.java
UsingException.java
UsingException1.java
UsingException2.java
UsingException
doesNotThrowException
main (String args[])
throwException ()
// UsingExceptions.java
// Demonstration of the try-catch-finally
// exception handling mechanism.
public class UsingException {
public static void main(String args[])
{
try {
throwException();
}
catch (Exception e)
{
System.err.println("Exception handled in main");
}
doesNotThrowException();
}
public static void throwException() throws Exception
{
// Throw an exception and immediately catch it.
try {
System.out.println("Method throwException");
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Method throwException
Exception handled in method throwException
Finally executed in throwException
Exception handled in main
Method doesNotThrowException
Finally executed in doesNotThrowException
End of method doesNotThrowException
Press any key to continue...
For Help, press F1
Ln 20, Col 56
DOS
Start
programming 3
chapter14 - JCreator - ...
C:\Program Files\Xi...
8:49 PM

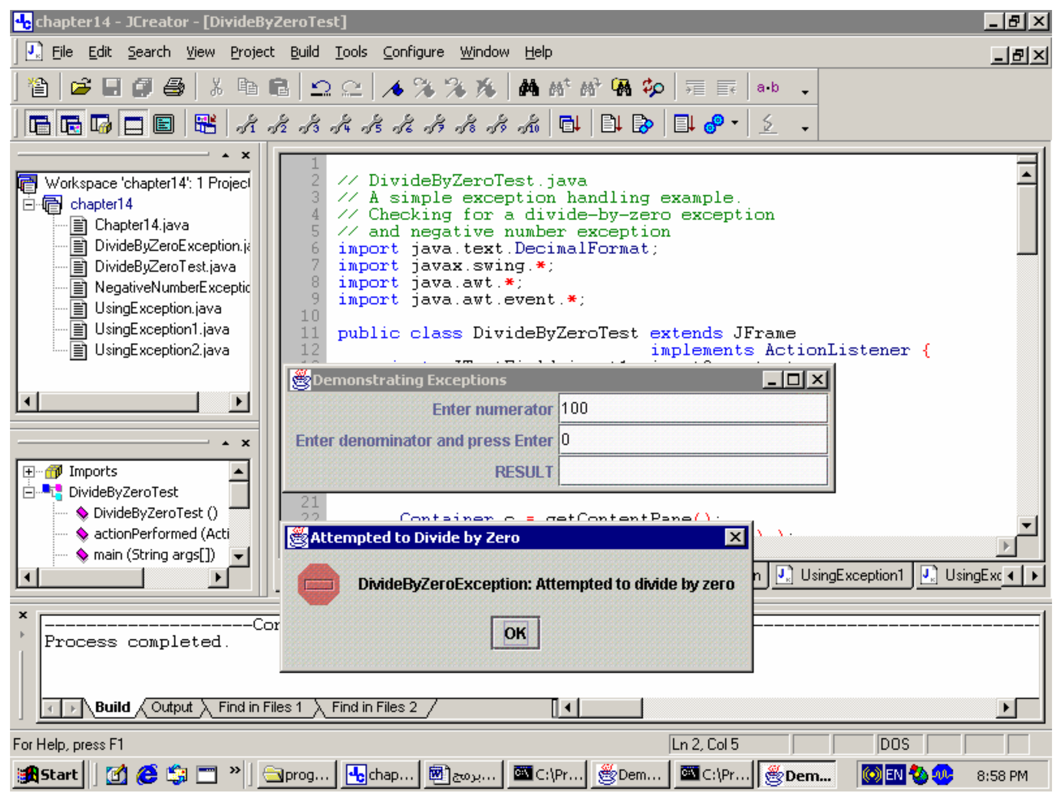


chapter14 - JCreator - [UsingException1]
File Edit Search View Project Build Tools Configure Window Help
Workspace 'chapter14': 1 Project
chapter14
Chapter14.java
DivideByZeroTest.java
UsingException.java
UsingException1.java
UsingException2.java
UsingException1
main (String args[])
throwException ()
// UsingExceptions.java
// Demonstration of stack unwinding.
public class UsingException1 {
public static void main(String args[])
{
try {
throwException();
}
catch (Exception e) {
System.err.println("Exception handled in main");
}
}
public static void throwException() throws Exception
{
// Throw an exception and catch it in main.
try {
System.out.println("Method throwException");
throw new Exception(); // generate exception
}
catch(RuntimeException e) { // nothing caught here
System.err.println("Exception handled in " +
DivideByZeroTest
NegativeNumberException
UsingException
UsingException1
C:\Program Files\Xinox Software\JCreator\GE2001.exe
Method throwException
Finally is always executed
Exception handled in main
Press any key to continue...
For Help, press F1
Ln 1, Col 7
DOS
Start
8:51 PM






Demonstrating Exceptions
Enter numerator 100
Enter denominator and press Enter -1
RESULT
Attempted to Enter negative numbers
NegativeNumberException: Attempted to Enter negative numbers
OK


Demonstrating Exceptions
Enter numerator 100
Enter denominator and press Enter 0
RESULT
Attempted to Divide by Zero
DivideByZeroException: Attempted to divide by zero
OK

DivideByZeroException.java
Created with JBuilder

```
// DivideByZeroException.java
// Definition of class DivideByZeroException.
// Used to throw an exception when a
// divide-by-zero is attempted.
 public class DivideByZeroException
          extends ArithmeticException {
   public DivideByZeroException()
   {
     super( "Attempted to divide by zero" );
   }

   public DivideByZeroException( String message )
   {
     super( message );
   }
```

NegativeNumberException.java
Created with JBuilder

```
// NegativeNumberException.java
// Definition of class NegativeNumberException.
// Used to throw an exception when a
// negative number is intered
 public class NegativeNumberException
          extends ArithmeticException {
   public NegativeNumberException()
   {
     super( "Attempted to Enter negative numbers" );
   }

}
```

DivideByZeroTest.java
Created with JBuilder

```
// DivideByZeroTest.java
// A simple exception handling example.
// Checking for a divide-by-zero exception
// and negative number exception
import java.text.DecimalFormat;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DivideByZeroTest extends JFrame
                    implements ActionListener {
  private JTextField input1, input2, output;
  private int number1, number2;
  private double result;

  // Initialization
  public DivideByZeroTest()
  {
    super( "Demonstrating Exceptions" );

    Container c = getContentPane();
    c.setLayout( new GridLayout( 3, 2 ) );

    c.add( new JLabel( "Enter numerator ",
                 SwingConstants.RIGHT ) );
    input1 = new JTextField( 10 );
    c.add( input1 );

    c.add(
      new JLabel( "Enter denominator and press Enter ",
               SwingConstants.RIGHT ) );
    input2 = new JTextField( 10 );
    c.add( input2 );
    input2.addActionListener( this );

    c.add( new JLabel( "RESULT ", SwingConstants.RIGHT ) );
    output = new JTextField();
    c.add( output );

    setSize( 425, 100 );
    show();
  }
```

```
// Process GUI events
public void actionPerformed( ActionEvent e )
{
  DecimalFormat precision3 = new DecimalFormat( "0.000" );

  output.setText( "" ); // empty the output JTextField

  try {
    number1 = Integer.parseInt( input1.getText() );
    number2 = Integer.parseInt( input2.getText() );
                    if(number1 < 0 || number2 <0)
                     throw new NegativeNumberException();
    result = quotient( number1, number2 );
    output.setText( precision3.format( result ) );
  }
  catch ( NumberFormatException nfe ) {
    JOptionPane.showMessageDialog( this,
      "You must enter two integers",
      "Invalid Number Format",
      JOptionPane.ERROR_MESSAGE );
  }
  catch ( DivideByZeroException dbze ) {
    JOptionPane.showMessageDialog( this, dbze.toString(),
      "Attempted to Divide by Zero",
      JOptionPane.ERROR_MESSAGE );
  }
  catch ( NegativeNumberException nne ) {
    JOptionPane.showMessageDialog( this, nne.toString(),
      "Attempted to Enter negative numbers",
      JOptionPane.ERROR_MESSAGE );
  }
}

// Definition of method quotient. Used to demonstrate
// throwing an exception when a divide-by-zero error
// is encountered.
public double quotient( int numerator, int denominator )
  throws DivideByZeroException
{
  if ( denominator == 0 )
    throw new DivideByZeroException();

  return ( double ) numerator / denominator;
}

public static void main( String args[] )
{
  DivideByZeroTest app = new DivideByZeroTest();
```

```
      app.addWindowListener(
        new WindowAdapter() {
          public void windowClosing( WindowEvent e )
          {
            e.getWindow().dispose();
            System.exit( 0 );
          }
        }
      );
    }
  }
```

UsingException.java
Created with JBuilder

```
// UsingExceptions.java
// Demonstration of the try-catch-finally
// exception handling mechanism.
public class UsingException {
  public static void main( String args[] )
  {
    try {
      throwException();
    }
    catch ( Exception e )
    {
      System.err.println( "Exception handled in main" );
    }

    doesNotThrowException();
  }
  public static void throwException() throws Exception
  {
    // Throw an exception and immediately catch it.
    try {
      System.out.println( "Method throwException" );
      throw new Exception();  // generate exception
    }
    catch( Exception e )
    {
      System.err.println(
        "Exception handled in method throwException" );
      throw e;  // rethrow e for further processing

      // any code here would not be reached
    }
```

```
    finally {
      System.err.println(
        "Finally executed in throwException" );
    }

    // any code here would not be reached
  }
  public static void doesNotThrowException()
  {
    try {
      System.out.println( "Method doesNotThrowException" );
    }
    catch( Exception e )
    {
      System.err.println( e.toString() );
    }
    finally {
      System.err.println(
        "Finally executed in doesNotThrowException" );
    }
    System.out.println(
      "End of method doesNotThrowException" );
  }
}
```

UsingException1.java

Created with JBuilder

```
// UsingExceptions.java
// Demonstration of stack unwinding.
public class UsingException1 {
  public static void main( String args[] )
  {
    try {
      throwException();
    }
    catch ( Exception e ) {
      System.err.println( "Exception handled in main" );
    }
  }

  public static void throwException() throws Exception
  {
    // Throw an exception and catch it in main.
    try {
      System.out.println( "Method throwException" );
      throw new Exception();      // generate exception
    }
    catch( RuntimeException e ) {  // nothing caught here
```

```
      System.err.println( "Exception handled in " +
                          "method throwException" );
    }
    finally {
      System.err.println( "Finally is always executed" );
    }
  }
}
```

```
UsingException2.java
Created with JBuilder
// UsingExceptions.java
// Demonstrating the getMessage and printStackTrace
// methods inherited into all exception classes.
public class UsingException2 {
  public static void main( String args[] )
  {
    try {
      method1();
    }
    catch ( Exception e ) {
      System.err.println( e.getMessage() + "\n" );
      e.printStackTrace();
    }
  }

  public static void method1() throws Exception
  {
    method2();
  }

  public static void method2() throws Exception
  {
    method3();
  }

  public static void method3() throws Exception
  {
    throw new Exception( "Exception thrown in method3" );
  }
}
```



برمجة ٣

معالجة الحدث

معالجة الحدث

٣

الجدارة:

أن يكون المتدرب قادراً على فهم أساسيات تصميم واجهات المستخدم، والقدرةعلى كتابة برامج لبناء مثل هذه الواجهات، واستخدام الأجزاء الرسومية المختلفة

الأهـداف:

بنهاية هذه الوحده، عليك ان تكون قادرا على كتابة برنامج جافا لتم بما يلي:

٧. فهم أساسيات تصميم الواجهات الرسومية.
٨. بناء واجهات التطبيق الرسومية
٩. انشاء ومعالجة الأجزاء الرسويةالمختلفة

مستوى الأاء المطلوب:

أ يصل المتدرب إلى إقان الجدارةبنسبة ١٠٠٪

الوسائل المساعدة

- وجود حاسب آي
- دفتر
- قلم

معالجة الحدث
Event Handling

مقدمة

إخال البيانات في البرامج التطبيقية التي تنفذ من خط الأوامر command line أو لوحة المراقبة console application دائما تكون تحت سيطرة البرنامج حيث يدخل المستخدم البيانات بترتيب معين ولكن البرامج التي تستخدمها يوميا في حاسبك لاتعمل بهذا الألوب وفيها يتم استخدام واجهة المستخدم الرسومية (GUI) graphical user interface ويكون التحكم لدى المستخدم حيث يمكنه استخدام كلاً من الفأرة ولوحة المفاتيح والتعامل مع واجهة المستخدم بأى ترتيب يريده ومثال ذلك يمكن للمستخدم إخال معلومات في مجال نصيText field أو اختيار من قائمة pull down menu أو الضغط على مفتاح click button أو إغلاق نافذةclose window والبرنامج يجب أن يتفاعل مع أوامر المستخدم بأي ترتيب تصل اليه والتعامل مع العديد من المدخلات الممكنة بترتيب عشوائي أمر صعب عند إلزام المستخدم ادخال البيانات بترتيب ثابت. في هذه الوحدة سوف نتعلم كيف نجعل برنامج جافا يستطيع أن يتفاعل مع أحداث واجهة المستخدم

الحدث Event الإنصات للحدث Event listeners ومصادر الحدث Event Sources

عندما يكتب مستخدم البرامج الرسومية مجموعة حروف أو يستخدم الفأرة في أي مكان داخل نافذة البرنامج يرسل مدير النافذة في الجافا java window manager إشعاراً للبرنامج أنه وقع حدث event وللعلم أن مدير النافذة يولد العديد من الأحداث events مثال ذلك عند تحريك الفأرة فترة قصيرة جداً على النافذة يولد حدث mouse event والكثير من البرامج لا تهتم بجميع الأحداث التي يولدها مدير النافذة ولذلك يجب على كل برنامج إيضاح أي الأحداث يحب استقبالها ويتم ذلك بإضافة كائنات استماع للحدث event listener وعلأوة على ذلك يوجد العديد من أنواع الأحداث مثال ذلك:

- أحداث لوحة المفاتيح Keyboard events
- أحداث حركة الفأرة mouse move events
- أحداث الضغط على الفأرة mouse click events
- أحداث غلق النافذة Window close events
- أحداث الضغط على زر Button click events

ولتجعل الاستماع للحدث أكثر تنظيماً تستخدم فصائل الاستماع للحدث event listener classes للاستماع للأنواع المختلفة من الأحداث.

ولتثبيت مستمع للحدث event listener يجب أن تعرف مصدر الحدث event source ومصدر الحدث هو مكونة واجهة المستخدم user interface component التي تولد حدثاً معيناً مثال ذلك:

- الزر button هو مصدر الحدث لإحداث الضغط على الزر Button click events
- القائمة menu هي مصدر حدث اختيار القائمة menu selection event
- شريط الزلاقة scrollbar هو مصدر حدث ضبط شريط الزلاقة scrollbar adjustment event

ويجب أن تخبر مصدر الحدث event source أى مستمع للأحداث تريد تثبيته.

مثال:

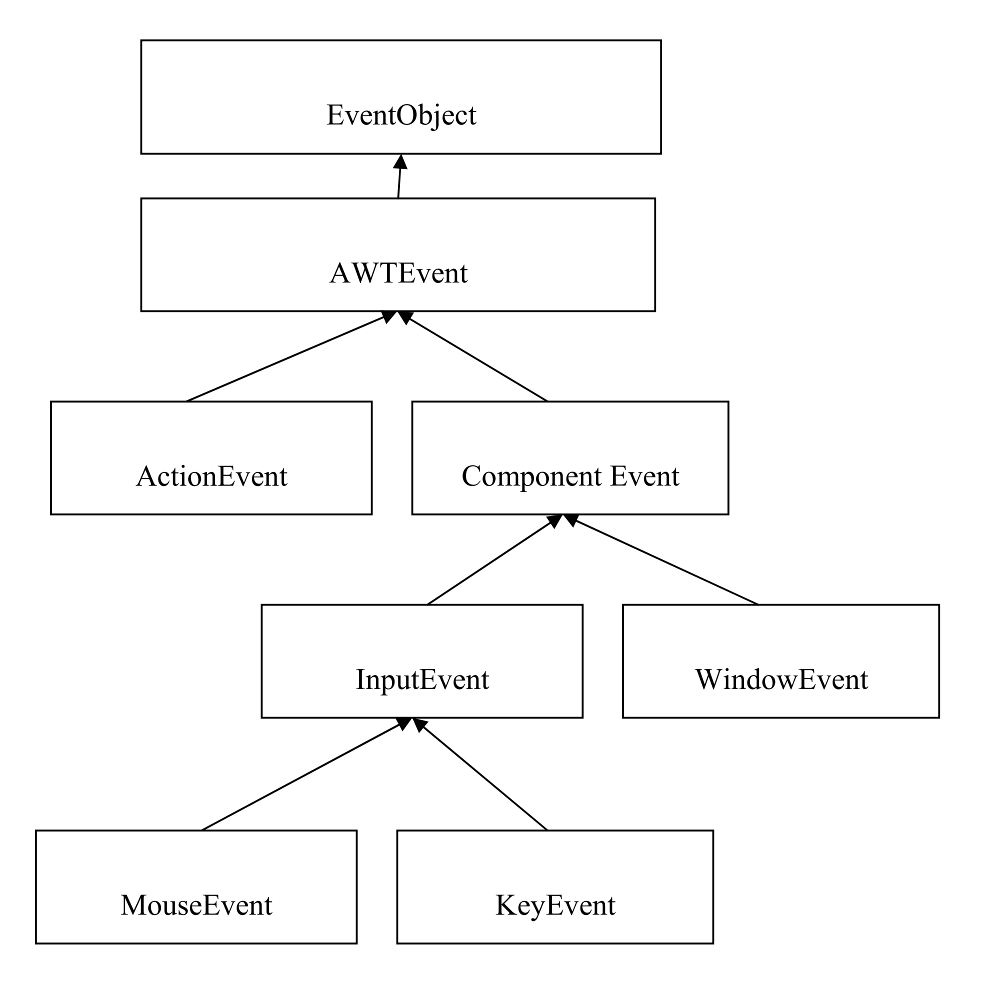
في هذا المثال سوف نستمع لحدث الضغط على الفأرة في برمجيات an applet ويوجد ثلاث فصائل تشمل هذا الحدث:

١ - **فصيلة الحدث The event class** في حالة الضغط على الفأرة توجد الفصيلة MouseEvent حيث إن الكائن المنشيء من هذه الفصيلة يخبر عن موضع مؤشر الفأرة (x و y) وما هو زر الفأرة الذى ضغطه المستخدم

٢ - **فصيلة المستمع The listener class** توجد الفصيلة MouseListener التي تستخدم لتنفيذ واجهة الاستماع للفأرة ولها العديد من الطرق التي تستدعى عند الضغط على زر الفأرة أو عند الرفع من على زر الفأرة ولكل هذه الطرق عوامل للفصيلة MouseEvent

٣ - **مصدر الحدث The event source** هذه هي المكونة component التي تولد حدث الفأرة وتدير الاستماع وفي هذا المثال يكون مصدر الحدث هو البرمجيات the applet التي ربما يضغط المستخدم الفأرة على مساحة سطحها وهنا يجب أن نخبرها ما هو مستمع الفأرة يجب اشعارها به عند وقوع حدث الفأرة.

كل فصائل الحدث event classes هي فصائل فرعية من الفصيلة EventObject ويبين شكل (٣ -١) مخطط الوراثة لمعظم فصائل الحدث. الفصيلة EventObject تمتلك الطريقة المهمة getSource التي تسترجع الكائن الذى يولد الحدث والفصائل الفرعية تمتلك الطرق الخاصة بها التي تصف الحدث أكثر ومثال ذلك الفصيلة MouseEvent تمتلك الطريقة getX و الطريقة getY التي تخبر عن موقع الفأرة أثناء توليد الحدث.



شكل (٣ -١)

الجزء الأكثر تعقيداً في معالجة الحدث في لغة الجافا هو الاقتراب من المستمع ومستمع الفأرة يجب أن ينفذ الخمس طرق التالية من الرابط MouseListener interface

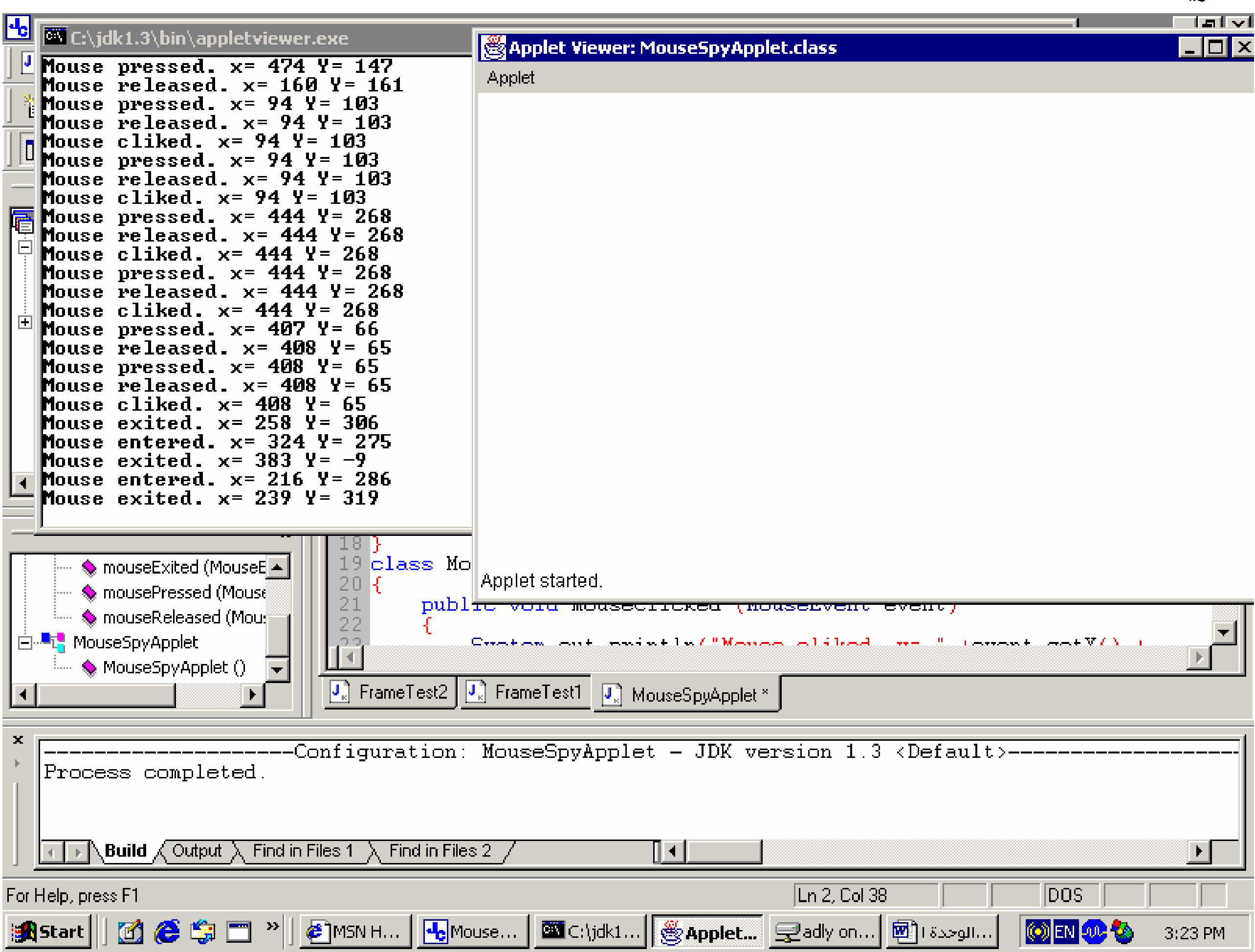
```
public interface Mouselistener
{
        void mouseClicked (MouseEvent event)
                // called when the mouse has been clicked on a component
        void mouseEntered (MouseEvent event)
                // called when the mouse enters a component
        void mouseExited (MouseEvent event)
                // called when the mouse exits a component
        void mousePressed (MouseEvent event)
                // called when the mouse button has been presseded on a component
        void mouseReleaseded (MouseEvent event)
                // called when the mouse button has been releaseded on a component
}
```

والآن نريد أن نراقب أحداث الفأرة ونظهرها كما تحدث ولهذا الغرض تم إنشاء الفصيلة MouseSpy لننفذ طرق المستمع في الفصيلة Mouselistener بحيث نظهر سبب الحدث وموضع الفأرة (X و Y) ويبين شكل (٣ -٢) برنامجاً لتنفيذ هذه الفصيلة والطرق جميعها في الأسطر من ١٧ إلى ٤٤ والآن نريد أن نثبت المستمع ولذلك تم استدعاء الطريقة addMouseListener لمصدر الحدث وهو applet وتم إنشاء كائن من الفصيلة MouseSpy وتمريره كمعامل للطريقة addMouseListener كما هو مبين في البرنامج في الأسطر ١٣ و ١٤

```
1.  //listen to mouse evnts in an applet
2.  import java.awt.*;
3.  import java.applet.*;
4.  import java.awt.event.MouseListener;
5.  import java.awt.event.MouseEvent;
6.  public class MouseSpyApplet extends Applet
7.  {
8.  public MouseSpyApplet()
9.    {
10.     MouseSpy listener = new MouseSpy();
11.     addMouseListener(listener);
12.   }
13. }
14. class MouseSpy implements MouseListener
15. {
16. public void mouseClicked (MouseEvent event)
17.   {
18.        System.out.println("Mouse cliked. x= " +event.getX() +
19.        " Y= " +event.getY());
20.   }
21. public void mouseEntered (MouseEvent event)
22.   {
23.     System.out.println("Mouse entered. x= " +event.getX() +
24.     " Y= " +event.getY());
25.   }
26. public void mouseExited (MouseEvent event)
27.   {
28.      System.out.println("Mouse exited. x= " +event.getX() +
29.      " Y= " +event.getY());
30.   }
31. public void mousePressed (MouseEvent event)
32.   {
33.     System.out.println("Mouse pressed. x= " +event.getX() +
34.     " Y= " +event.getY());
35.   }
36. public void mouseReleased (MouseEvent event)
37.   {
38.     System.out.println("Mouse released. x= " +event.getX() +
39.     " Y= " +event.getY());
40.   }
41. }
```

شكل (٣-٢)

ولمتابعة هذا البرنامج والاستفادة منه يجب تنفيذه ويكون ناتج هذا البرنامج كما هو مبين في شكل (٣- ٣) انه كلما أحست البرمجيات applet the بحدث الفأرة تستدعى الطريقة المناسبة لكائن المستمع كمثال لذلك في السطر الأول من شكل (٣ -٣) نجد أن المستخدم ضغط على زر الفأرة في الموقع x=474 و y=147 فتم استدعاء الطريقة mousePressed والسطر الثاني يبين أن المستخدم رفع يده عن زر الفأرة في الموقع x=168 و y=161 فتم استدعاء الطريقة mouseReleased والسطر الثالث والرابع والخامس يبين أن المستخدم ضغط ورفع بسرعة عند الموقع x=94 و y=103 ولذلك تم استدعاء الطرق mousePressed و mouseReleased و mouseClicked والسطر قبل الأخير من الشكل يبين أن الفأرة دخلت سطح البرمجيات applet في الموقع x=216 و y=286 ولذلك تم استدعاء الطريقة mouseEntered وخرجت الفأرة من سطح البرمجيات في الموقع x=239 و y=319 ولذلك تم استدعاء الطريقة mouseExited.



شكل(٣ -٣)

مهايأة الحدث Event Adapter

في الجزء السابق رأينا كيفية تثبيت مستمع listener لمصدر حدث الفأرة وكيفية استدعاء طرق المستمع listener methods عند وقوع الحدث . وعادة البرنامج لايهتم بكل إشعارات المستمع ومثال ذلك أن البرنامج يريد الاستماع فقط للضغط السريع على زر الفأرة mouse click ولايهتم أن الضغط السريع يتكون من حدث الضغط على زر الفأرة mouse pressed و الرفع من على زر الفأرة mouse released وبالطبع يمكن للبرنامج إعطاء المستمع الذى يعرف جميع الطرق methods التي لايهتم بها البرنامج أن لاتفعل شيئاً كالتالي:

```
1.  class MouseClickListener implements MouseListener
2.  {
3.  public void mouseClicked (MouseEvent event)
4.    {
5.      // mouse click action here
6.    }

7.  public void mouseEntered (MouseEvent event)
8.      {
9.          // do nothing
10.   }
11. public void mouseExited (MouseEvent event)
12.   {
13.       // do nothing
14.   }
15. public void mousePressed (MouseEvent event)
16.   {
17.      // do nothing
18.
19.   }
20. public void mouseReleased (MouseEvent event)
21.   {
22.     // do nothing
23.
24.   }
25. }
```

ويوجد في لغة الجافا فصيلة مهيء adapter class الذى ينفذ الرابط MouseListener interface حيث إن جميع الطرق methods لاتفعل شيئاً do nothing كالتالي

```
1.  class MouseAdapter implements MouseListener
2.  // This class is defined in the java.awt.event package
3.  {
4.  public void mouseClicked (MouseEvent event)
5.    {
6.               // do nothing
7.    }
8.  public void mouseEntered (MouseEvent event)
9.      {
```

```
10.         // do nothing
11.   }
12. public void mouseExited (MouseEvent event)
13.   {
14.       // do nothing
15.   }
16. public void mousePressed (MouseEvent event)
17.   {
18.      // do nothing
19.   }
20. public void mouseReleased (MouseEvent event)
21.   {
22.     // do nothing
23.   }
24. }
```

وبذلك يمكنك تعريف فصيلة مستمع للضغط على الفأرة ترث .الفصيلة MouseAddapter وتكتب فقط الطرق التي تهتم بها فقط كالتإلي

```
class MouseClickListener extends MouseAdapter
{
        public void mouseClicked (MouseEvent event)
         {
                 // mouse click action here
         }
```

العديد من واجهات المستمع listener interface مثل واجهة مستمع الفعل ActionListener تمتلك طريقة واحدة فقط وفي هذه الحالة لايوجد مهيء مطابق حيث يمكن تنفيذ الواجهة implement interface بنفس سهولة وراثة (امتداد) المهيء extend the adapter ولكن الحزمة java.awt.event تحتوي على فصيلة مهيءadapter class لجميع واجهات مستمع الحدث التي تمتلك على الأقل طريقتين two methods ومثال ذلك مستمع النافذة WindowListener واجهة تحتوي على ٧ طرق ولذلك يوجد مهيء مطابق ينفذ الطرق السبعة بحيث لاتفعل شيئاً وسوف نستخدمه في هذه الوحدة لاحقاَ

تنفيذ المستمع كفصيلة داخلية Implementing Listener as inner class

في المثال السابق مستمع الفأرة كان يطبع كل أحداث الفأرة باستخدام System.out والأن نفترض أنك تريد شيئاً أكثر جاذبية يحدث عندما يضغط المستمع على الفأرة سريعاً وهنا في شكل (٣ -٤) برنامج يرسم قطع ناقص ellipse على الشاشة ويحرك القطع الناقص إلى موضع ضغط زر الفأرة.

إذا أردنا رسم قطع ناقص فقط يكون البرنامج بالشكل الآتي:

```
import java.awt.*;
```

```
import java.applet.*;
import java.awt.event.*;
import java.awt.geom.*;

public class EggApplet extends Applet
{
        public EggApplet()
        {
                egg = new Ellipse2D.Double(0,0,EGG_WIDTH, EGG_HIGHT);
        }
        public void paint(Graphics g)
        {
                Graphics2D g2 = (Graphics2D)g;
                g2.draw(egg);
        }
        private Ellipse2D.Double egg;
        private static final double EGG_WIDTH = 30;
        private static final double EGG_HIGHT = 50;
```

والآن دعنا نضيف مستمع للفأرة ونحاول تحريك القطع الناقص إلى موضع الفأرة ويصبح البرنامج كالتالي:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.awt.geom.*;

public class EggApplet extends Applet
{
        public EggApplet()
        {
                egg = new Ellipse2D.Double(0,0,EGG_WIDTH, EGG_HIGHT);

                // add mouse click listener
                MouseClickListener listener = new MouseClickListener();
                addMouseListener(listener);
        }
        public void paint(Graphics g)
        {
                Graphics2D g2 = (Graphics2D)g;
                g2.draw(egg);
        }
        private Ellipse2D.Double egg;
        private static final double EGG_WIDTH = 30;
        private static final double EGG_HIGHT = 50;

        class MouseClickListener extends MouseAdapter
        {
                public void mouseClicked(MouseEvent event)
```

```
int mouseX=event.getX();
int mouseY=event.getY();

// now move the ellipse to (mouseX, MouseY)

egg.setFrame(mouseX-EGG_WIDTH/2, mouseY-EGG_HIGHT/2,
EGG_WIDTH, EGG_HIGHT);
repaint();
```

عند ترجمة البرنامج نجد أنه يعطي الأخطاء التالية وذلك لأن الفصيلة MouseClickListener ليس لها الحق في استخدام متغيرات الكائن الخاصة بالفصيلة EggApplet.

```
C:\programming 3\chapter10\EggApplet\EggApplet.java:52: cannot resolve symbol
symbol  : variable EGG_WIDTH
location: class MouseClickListener
                        egg.setFrame(mouseX-EGG_WIDTH/2, mouseY-EGG_HIGHT/2,
                             ^
C:\programming 3\chapter10\EggApplet\EggApplet.java:52: cannot resolve symbol
symbol  : variable EGG_HIGHT
location: class MouseClickListener
                        egg.setFrame(mouseX-EGG_WIDTH/2, mouseY-EGG_HIGHT/2,
                                              ^
C:\programming 3\chapter10\EggApplet\EggApplet.java:53: cannot resolve symbol
symbol  : variable EGG_WIDTH
location: class MouseClickListener
                        EGG_WIDTH, EGG_HIGHT);
                 ^
C:\programming 3\chapter10\EggApplet\EggApplet.java:53: cannot resolve symbol
symbol  : variable EGG_HIGHT
location: class MouseClickListener
                        EGG_WIDTH, EGG_HIGHT);
                        ^
C:\programming 3\chapter10\EggApplet\EggApplet.java:52: cannot resolve symbol
symbol  : variable egg
location: class MouseClickListener
                        egg.setFrame(mouseX-EGG_WIDTH/2, mouseY-EGG_HIGHT/2,
                 ^
C:\programming 3\chapter10\EggApplet\EggApplet.java:54: cannot resolve symbol
symbol  : method repaint  ()
location: class MouseClickListener
                        repaint();
                 ^
6 errors
Process completed.
```

وهذه الحالة قياسية لمستمع الحدث event listener لأنه عادة تحتاج طرق الحدث للوصول إلى متغيرات في فصيلة آخرى وذلك يمكن علاجه باستخدام المستمع كفصيلة داخلية inner class

والفصيلة الداخلية تعرف داخل فصيلة آخرى وتتعامل كأنها فصيلة عادية حيث يمكن إنشاء كائنات والتفاعل مع الطرق بالأساليب المعتادة ولكن هناك استثناء حيث إن الطرق في الفصيلة الداخلية يسمح لها باستخدام متغيرات الكائن instance variables وطرق الفصيلة الخارجية وتكون الصيغة العامة في لغة الجافا كالتالي:

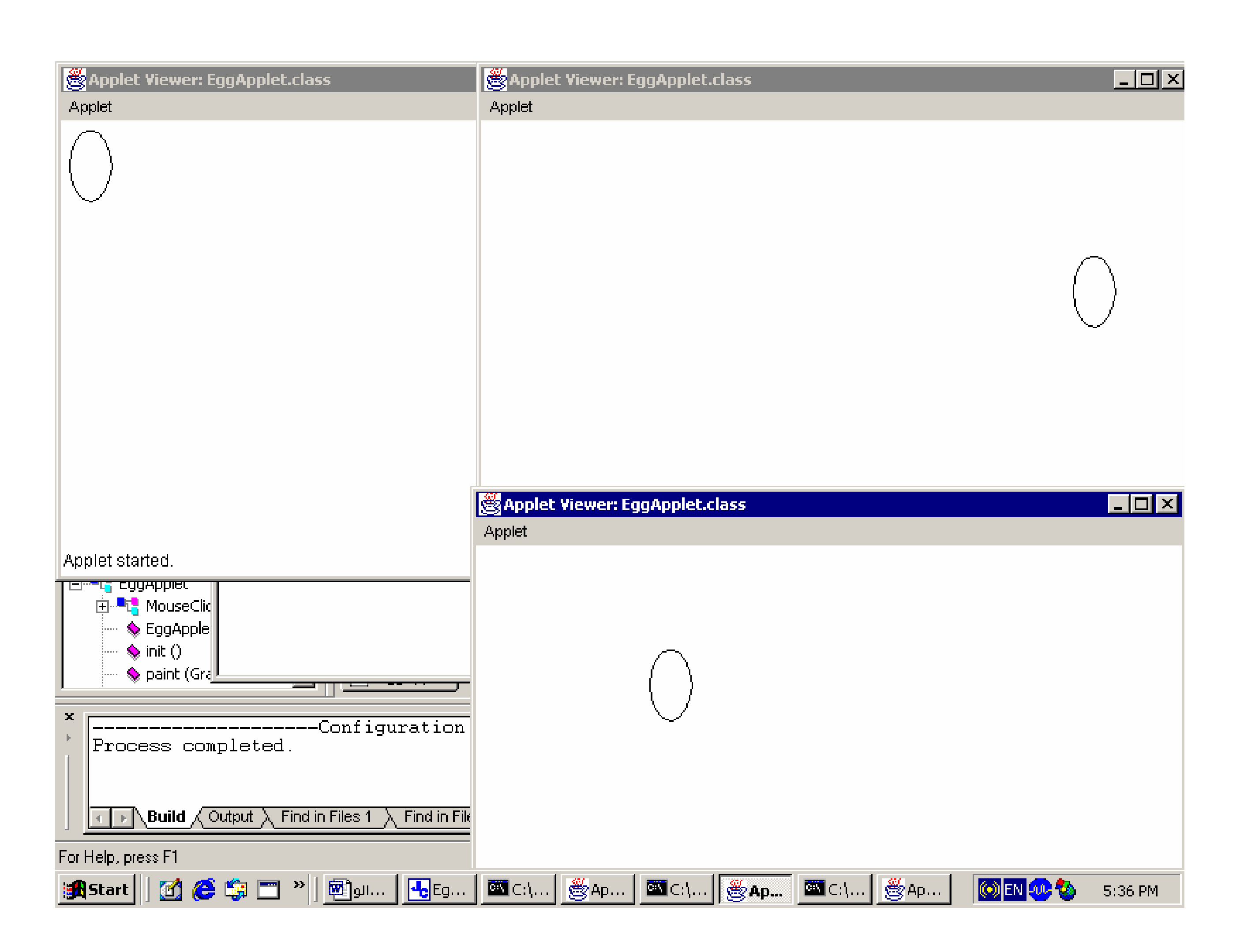
```
Class   OuterClassName
{
        .........
        accessSpecifier          class    InnerClassName
        {
                methods
                variables
        }
```

شكل (٣ -٤) يمثل البرنامج حيث تم ترجمته بشكل صحيح وتشغيله كما هو مبين في شكل (٣ -٥)

```
1.  // using inner class
2.  import java.awt.*;
3.  import java.applet.*;
4.  import java.awt.event.*;
5.  import java.awt.geom.*;
6.  public class EggApplet extends Applet
7.  {
8.  public EggApplet()
9.  {
10. egg = new Ellipse2D.Double(0,0,EGG_WIDTH, EGG_HIGHT);
11. // add mouse click listener
12. MouseClickListener listener = new MouseClickListener();
13. addMouseListener(listener);
14. }
15. public void paint(Graphics g)
16. {
17. Graphics2D g2 = (Graphics2D)g;
18. g2.draw(egg);
19. }
20. private Ellipse2D.Double egg;
21. private static final double EGG_WIDTH = 30;
22. private static final double EGG_HIGHT = 50;
23. // inner class definition
24.
25. private class MouseClickListener extends MouseAdapter
26. {
```

```
27. public void mouseClicked(MouseEvent event)
28. {
29. int mouseX=event.getX();
30. int mouseY=event.getY();
31. // now move the ellipse to (mouseX, MouseY)
32. egg.setFrame(mouseX-EGG_WIDTH/2, mouseY-EGG_HIGHT/2,
33. EGG_WIDTH, EGG_HIGHT);
34. repaint();
35. }
36. }
37. }
```

شكل(٣ -٤)



شكل(٣ -٥)

نوافذ الإطار Frame Windows

جميع البرامج الرسومية التي كتبت سابقا استخدمت البرمجيات applets ويمكنك الآن تعلم كتابة برامج رسومية من خلال تطبيقات الجافا java applications وكل برنامج رسومي يستخدم نافذة إطار frame window أو أكثر ونافذة الإطار لها شريط عنوان title bar و حد border

لكي تظهر الإطار نستخدم الفصيلة JFrame من الحزمة javax.swing ويجب تحديد مقاس الإطار باستدعاء الطريقة setSize ولتحديد عنوان الإطار نستخدم الطريقة setTitle والطريقة show تجعل مدير النافذة window manager يظهر الإطار ويبين شكل (٣ -٦) برنامج بسيط لإظهار إطار نافذة بدون أي شيء داخله وكما تعلمنا سابقاً أن تطبيقات الجافا لابد أن تحتوي على الطريقة الرئيسة main وفي السطر ٤ يتم إنشاء الكائن frame من الفصيلة EmptyFrame التي هي امتداد للفصيلةJFrame وفي منشيء الفصيلة EmptyFrame يتم تحديد حجم الإطار وذلك في الأسطر من ٩ إلى ١٧ وفي الطريقة main أيضاً يتم تحديد عنوان الإطار في السطر ٥ و إظهار الإطار في السطر ٦

```
1.  import javax.swing.JFrame;
2.  public class FrameTest1
3.  { public static void main (String[] args)
4.      { EmptyFrame frame = new EmptyFrame();
5.        frame.setTitle("frameTest");
6.        frame.show();
7.      }
8.  }
9.  class EmptyFrame extends JFrame
10. {
11. public EmptyFrame()
12.   {
13.    final int DEFAULT_FRAME_WIDTH =300;
14.    final int DEFAULT_FRAME_HIGHT =300;
15.    setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HIGHT);
16.   }
17. }
```

شكل (٣ -٦)

عند تنفيذ البرنامج يتم إظهار الإطار وينتهي تنفيذ الطريقة main ولكن يظل البرنامج يعمل ويبقى الإطار ظاهر على الشاشة ويمكن تحريكه وتغيير حجمه وهذا هو الفرق الأساسي بين البرامج التي تعمل من لوحة المراقبة console programs والبرامج الرسومية graphical programs

بمجرد إظهار نافذة الإطار يبدأ البرنامج خيطاً جديداً new thread لتنفيذ إظهار واجهة المستخدم الرسومية وعند انتهاء الطريقة main ينتهي خيط الطريقة main thread ولكن برنامج الجافا لاينتهي لأن خيط

واجهة المستخدم مازال يعمل وهذه مشكلة حيث إنه عند إغلاق نافذة الإطار بالضغط على أيقونة الغلق من شريط العنوان يظل البرنامج يشتغل ولايعمل شيئاً ولإنهاء البرنامج يجب استخدام الجملة التالية

```
System.exit(0)
```

والمشكلة أين تضع هذه الجملة. لايمكن وضعها في نهاية الطريقة main كالتالي

```
public class FrameTest1
        public static void main (String[] args)
    { EmptyFrame frame = new EmptyFrame();
      frame.setTitle("frameTest");
      frame.show();
      System.exit(0);    // Error
```

بهذا الوضع يظهر البرنامج النافذة للحظة وجيزة وينتهي فوراً وبدلاً عن ذلك نريد إنهاء البرنامج عندما يضغط المستخدم على أيقونة الغلق في شريط العنوان ولكن لانعرف متى يحدث ذلك لأن المستخدم هو المتحكم في البرنامج ويمكن عمل العديد من الأشياء بأي ترتيب والحل هو تثبيت معالج للحدث الذي يستدعى عندما يضغط المستخدم على أيقونة الغلق واستجابة لهذا الحدث ننهي البرنامج

ولتكتشف متى يغلق المستخدم النافذة يجب الاستماع لإحداث النافذة window events ويمكن حدوث سبعة أحداث للنافذة في برنامج الجافا كالتالي

١. النافذة فتحت الآن للمرة الأولى
٢. تم إغلاق النافذة نتيجة للطريقة dispose
٣. تم تنشيط النافذة بسبب الضغط داخلها
٤. تم إيقاف نشاط النافذة نتيجة للدخول في نافذة آخرى
٥. تم تحويل النافذة إلى أيقونة نتيجة الضغط على أيقونة التصغير في شريط العنوان
٦. تم استرجاع النافذة (من حالة أيقونة) نتيجة للضغط على أيقونة النافذة المصغرة
٧. تم غلق النافذة نتيجة الضغط على أيقونة الغلق في شريط العنوان

وللإستماع لهذه الأحداث يجب إضافة كائن مستمع النافذة window listener إلى الإطار وكائن مستمع النافذة ينفذ الرابط WindowListener الذي يمتلك سبع طرق كالتالي:

```
Public interface WindowListener

        void windowOpend(windowEvent e);
        void windowClosed(windowEvent e);
        void windowActivated(windowEvent e);
        void windowDeactivated(windowEvent e);
        void windowIconified(windowEvent e);
        void windowDeiconfied(windowEvent e);
```

void windowClosing(windowEvent e);

بالطبع البرنامج البسيط لايهتم بالأحداث الستة الأولى لأن الفصيلة العليا JFrame تستمع إليهم وتعرف كيف تتعامل معهم ولكن هناك برامج رسومية متخصصة تحتاج أن تعرف عن بعض هذه الأحداث مثال ذلك البرنامج الذي يظهر الرسوم المتحركة يريد أن يوقف الحركة عندما تصغر النافذة ويبدأ الحركة عند استعادة النافذة. ربما أن تتعجب لماذا لايستدعى كائن الفصيلة JFrame ببساطة الطريقة System.exit عندما يتم إغلاقه وذلك لسبب بسيط وهو أن البرنامج يمكن أن يمتلك العديد من نوافذ الإطار وليست فكرة طيبة أن ينتهي البرنامج كله إذا أغلق المستخدم إحدى نوافذه ولذلك يجب أن نعلم نافذة معينة أن تنتهي عندما يغلقها المستخدم ويتحقق ذلك بتثبيت كائن مستمع نافذة فيه الطريقة windowClosing لتنهي البرنامج ويمكن استخدام المهيء WindowAdapter الذى ينفذ طرق الرابط WindowListener بحيث لاتعمل شيئاً nothing do وهنا يتم إنشاء الفصيلة WindowCloser التي ترث الفصيلة WindowAdapterكالتإلى

class WindowCloser extends WindowAdapter

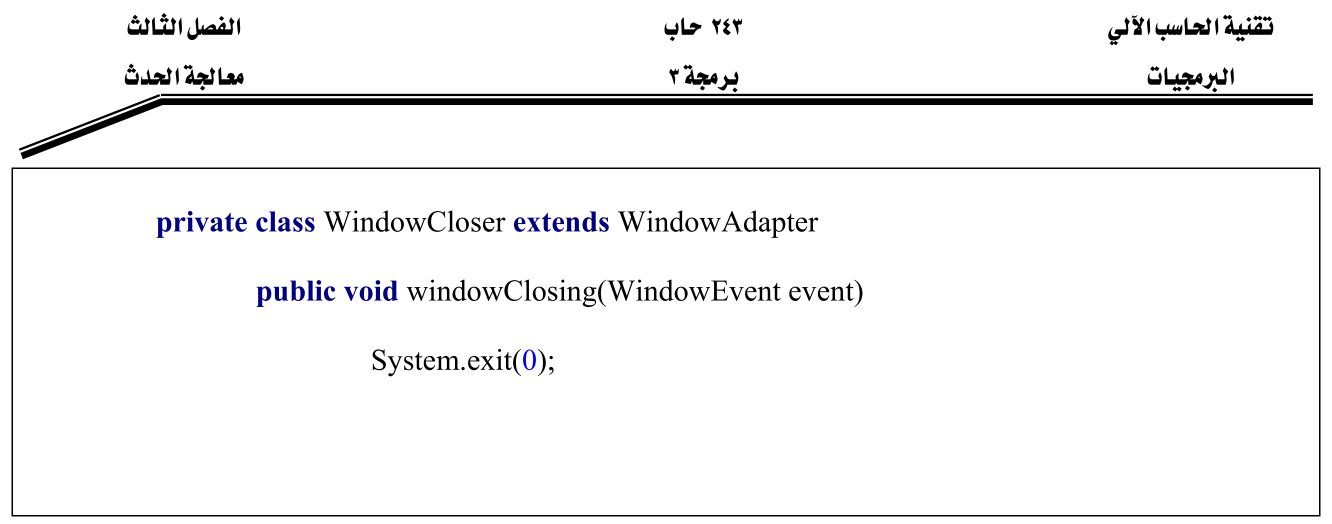
public void windowClosing(WindowEvent event)

System.exit(0);

وفي النهاية نحتاج أن نضيف كائن من الفصيلة WindowCloser كمستمع للنافذة للإطار باستدعاء الطريقة addWindowListener ويصبح البرنامج كما هو مبين في شكل (٣ -٧)

```
import javax.swing.JFrame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class FrameTest2
{         public static void main (String[] args)
          { EmptyFrame frame = new EmptyFrame();
                    frame.setTitle("Close me!");
                    frame.show();
          }
}
class EmptyFrame extends JFrame
{public EmptyFrame()
          {         final int DEFAULT_FRAME_WIDTH =300;
                    final int DEFAULT_FRAME_HIGHT =300;
                    setSize(DEFAULT_FRAME_WIDTH, DEFAULT_FRAME_HIGHT);

                     WindowCloser listener = new WindowCloser();
                     addWindowListener(listener1);
```



```
private class WindowCloser extends WindowAdapter
        public void windowClosing(WindowEvent event)
                System.exit(0);
```

شكل(٣ -٧)

والآن يمكن إنهاء التطبيق بصورة صحيحة عندما يغلق المستخدم نافذة الإطار

شكل(٣ -٩)



برمجة ٣

واجهات المستخدم الرسومية

الجدارة:

أن يكون المتدرب قادراً على فهم أساسيات تصميم واجهات المستخدم، والقدرة على كتابة برامج لبناء مثل هذه الواجهات، واستخدام الأجزاء الرسومية المختلفه.

الأهـداف:

بنهاية هذه الوحده، عليك أن تكون قادرا على:

١. فهم أساسيات تطبيقات الواجهات الرسومية.
٢. بناء واجهات التطبيق الرسومية
٣. إنشاء ومعالجة الأجزاء الرسومية المختلفة.
٤. كتابة برنامج جافا لدعم المفاهيم السابقة.

مستوى الاداء المطلوب:

أن يصل المتدرب إلى إتقان الجدارة بنسبة ١٠٠٪

الوسائل المساعده:

- وجود حاسب آلي يحتوي على بيئة متكاملة لكتابة برامج بلغة جافا
- دفتر
- قلم

واجهات المستخدم الرسومية

مقدمة :

يعتبر بناء واجهات المستخدم الرسومية من الأجزاء المهمة في البرنامج، حيث إن هذه الواجهات تعطي البرنامج شكلا معينا وشعورا معينا لدى المستخدم، حيث إن استخدام مفاهيم وأجزاء موحدة في بناء الواجهات للعديد من البرامج المختلفة يعطي المستخدم قدراً كبيراً من الراحة أثناء استخدام البرامج ويقلل من الوقت اللازم لتعلمها. هناك العديد من الواجهات المستخدمة التي قمت باستخدامها أثناء استعمالك للحاسب فشاشات نظام الويندوز والمستكشف وغيرها تستخدم واجهات مستخدم رسومية.

تتكون واجهة المستخدم الرسومية من العديد من المكونات components وهي عباره عن كائنات objects يستطيع المستخدم التعامل معها بواسطة الفأرة، لوحة المفاتيح وغيرها من الوسائل، والجدول التالي يوضح بعض المكونات الرسومية لواجهة التطبيق:

الوصف	اسم الجزء
مكان يوضع فيه نص أو صوره لا يمكن تغييره أو الكتابة عليه	JLabel
مكان يمكن أن يستقبل مدخلات من المستخدم وطباعة الناتج عليه.	JTextField
مكان يقوم بإطلاق حدث ما عند الضغط عليه	JButton
هو شكل رسومي يمكن أن يكون مختارا أو غير مختار	JCheckBox
هو عبارة عن قائمة من العناصر، يمكن للمستخدم الاختيار منها بالضغط على العنصر من القائمة.	JcomboBox
مكان يمكن لقائمة من العناصر أن تظهر فيه ليقوم المستخدم باختيار عنصر ما بالضغط عليه مرة بالفأرة.	JList
عباره عن حأوي لمجموعة العناصر الرسومية.	JPanel

مراجعه للحزمة swing

إن جميع الأجزاء الموضحة في الجدول أعلاه محتواه داخل الحزمة الرسومية المسماة javax.swing وقد أصبحت هذه الأجزاء الرسومية أساسية في لغة جافا في الإصدار Java 2 platform النسخه 1.2. كما أن معظم أجزاء الحزمة swing تم كتابتها ،و معالجتها وعرضها كليا بلغة جافا ، لذلك فهي تسمى Pure Java Components.

إن الأجزاء الرسومية الأصليه في جافا والموجودة في الحزمة AWT مرتبطة مباشرة مع الإمكانيات الرسومية للجهاز المستخدمة فيه، لذلك فإن أجزاء GUI ستظهر بشكل متباين على الاجهزه المختلفه وذلك لاختلاف platform على كل منها؛ عند كتابة برنامج يقوم برسم زر button على نظام التشغيل ويندوز فإن هذا الزر سيكون له نفس شكل الزر في نظام ويندوز، وعند رسمه على نظام Apple Macintosh سيكون له شكل الزر في نظام Apple Macintosh. لذلك وحيث إن الأجزاء الرسومية في الحزمة swing بلغة جافا أصلا، فإن الواجهات الرسومية التي تستخدم هذه الحزمة ستحافظ على شكلها ومظهرها حتى لو اختلف نظام التشغيل من جهاز إلى آخر. وسنستعرض الآن أهم عناصر ومكونات الحزمة swing

العنصر الرسومي JLabel

يوفر العنصر الرسومي JLabel تعليمات نصية أو معلومات في واجهة المستخدم الرسومية GUI. يعرف هذا العنصر الرسومي من خلال الفصيلة JLabel ويظهر سطر نصي واحد، أو صورة أو كلاهما، للقراءة فقط ولا يمكن التعديل عليه. المثال التالي سيعرفك على العديد من الطرق الخاصة بالعنصر الرسومي JLabel:

```
1
2     // Demonstrating the JLabel class.
3
4     // Java core packages
5     import java.awt.*;
6     import java.awt.event.*;
7
8     // Java extension packages
9     import javax.swing.*;
10
11    public class LabelTest extends JFrame {
12       private JLabel label1, label2, label3;
13
14       // set up GUI
15       public LabelTest()
16       {
17          super( "Testing JLabel" );
18
19          // get content pane and set its layout
20          Container container = getContentPane();
21          container.setLayout( new FlowLayout() );
22
23          // JLabel constructor with a string argument
```

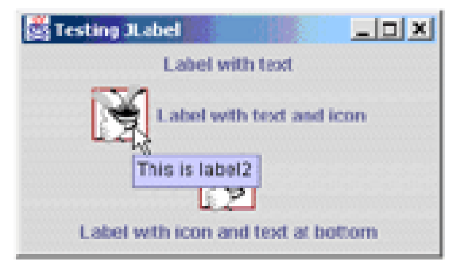
```
24          label1 = new JLabel( "Label with text" );
25          label1.setToolTipText( "This is label1" );

26          container.add( label1 );
27
28          // JLabel constructor with string, Icon and
29          // alignment arguments

30          Icon bug = new ImageIcon( "bug1.gif" );
31          label2 = new JLabel( "Label with text and icon",
32             bug, SwingConstants.LEFT );

33          label2.setToolTipText( "This is label2" );
34          container.add( label2 );
35
36          // JLabel constructor no arguments
37          label3 = new JLabel();
38          label3.setText( "Label with icon and text at bottom" );
39          label3.setIcon( bug );
40          abel3.setHorizontalTextPosition(SwingConstants.CENTER );
41          label3.setVerticalTextPosition( SwingConstants.BOTTOM );
42          label3.setToolTipText( "This is label3" );
43          container.add( label3 );
44
45          setSize( 275, 170 );
46          setVisible( true );
47       }
48
49       // execute application
50       public static void main( String args[] )
51       {
52          LabelTest application = new LabelTest();
53
54          application.setDefaultCloseOperation(
55             JFrame.EXIT_ON_CLOSE );
56       }
57
58    }  // end class LabelTest
```





يقوم البرنامج بتعريف ثلاثة عناصر (كائنات) من نوع JLabel في السطر ١٢. ثم قمنا بإعطاء العناصر قيم في الباني LabelTest Constructor (السطر ١٥ -٤٧). السطر ٢٤ ينشئ عنصر JLabel يحتوي على النص "Label with text". ويظهر هذا النص عند إظهار نافذة التنفيذ على الشاشه.

السطر ٢٥ يستخدم الطريقة setToolTipText لتحديد الملحظه التي ستظهر عند تحريك المؤشر فوق هذا العنصر.

السطر ٢٦ يضيف العنصر label1 إلى شريط المحتويات content pane. العديد من العناصر في الحزمة swing يمكنها إظهار صور وذلك بتحديد عنصر من نوع Icon كبارامیتر في منشيء العنصر أو باستخدام طريقة setIcon.

إحدى الفصائل التي تطبق الواجهة أيقون Icon interface هي الفصيله ImageIcon والتي تدعم العديد من تنسيقات الصور image formats مثل: GIF, PNG و GPEG. السطر ٣٠ يعرف الكائن من نوع ImageIcon. أما الملف bug1.gif فيحتوي على الصورة التي ستحمل وتخزن في الكائن ImageIcon، كما أننا نفترض أن ملف الصورة موجود في نفس الدليل folder الذي يحتوي على البرنامج. ان الكائن ImageIcon تم إسناده إلى متغير مرجعي من نوع أيقون Icon اسمه bug. تذكر إن الفصيلة ImageIcon تطبق الواجهة Icon وبالتالي فإن ImageIcon هو Icon.

الفصيلة JLabel تدعم عملية عرض الأيقونات، السطر ٣١ -٣٢ يستخدم منشئاً آخر للفصيل JLabel لإظهار label يعرض النص "Label with text and Icon" والأيقونة التي يشير إليها المتغير bug محدد اتجاه ليكون على الطرف اليسار من Label.حيث تم تحديد الاتجاه باستخدام ثابت عددي من خلال المعرف SwingConstants.LEFT. لاحظ أن الوضع التلقائي في حالة وجود نص مع أيقونة على نفس العنصر الرسومي label أن تكون الأيقونة على يمين النص. كما يمكنك تحديد التنسيقات الأفقية والعمودية للعنصر الرسومي label من خلال الطرق setHorizontalAlignment و setVerticalAlignment. السطر ٣٣ يحدد الملحظة tool tip للعنصر label2. والسطر ٣٤ يضيف هذا العنصر إلى content pane. السطر ٣٧ ينشئ كائناً من نوع label باستخدام باني من غير أي بارميتر وفي هذه الحالة فإن هذا الكائن لا يحتوي على نص أو أيقونة بداخله. السطر ٣٨ يستخدم الطريقة setText لتحديد النص للكائن label3 كما ويوجد هناك طريقة أخرى لاسترجاع النص من label وهي getText. السطر ٣٩ يستخدم الطريقة setIcon لتحديد الأيقونة للكائن label3 كما وتوجد هناك طريقه لاسترجاع الأيقونة الموجودة في الكائن وهي getIcon. الأسطر ٤٠ -٤١ تستخدم الطرق setHorizontalTextPosition و setVerticalTextPosition لتحديد موقع النص في

العنصر الرسومي label. ففي هذا البرنامج سيكون النص أفقيا في الوسط وعموديا في الأسفل بمعنى أن الأيقونة ستكون في الأعلى.

السطر ٤٢ يحدد الملحظه للكائن الرسومي label3 والسطر ٤٣ يضيف label3 إلى شريط المحتويات content pane.

العنصر الرسومي JTextField **و العنصر الرسومي** JPasswordField.

العنصران JTextField و JPasswordField هي مناطق أحادية السطر تستخدم لإدخال نص من قبل المستخدم عن طريق لوحة المفاتيح. في حين أن العنصر JPasswordField يظهر أن أحرف قد تم إدخالها دون إظهار الأحرف نفسها. عندما يدخل المستخدم البيانات في أحد هذين العنصرين ثم ضغط enter يتسبب ذلك في إطلاق حدث وفي حالة تسجيل أحد العنصرين أو كلاهما في مستمع للحدث event listener فانه يمكن معالجة الحدث واستخدام النص من العنصر الرسومي.

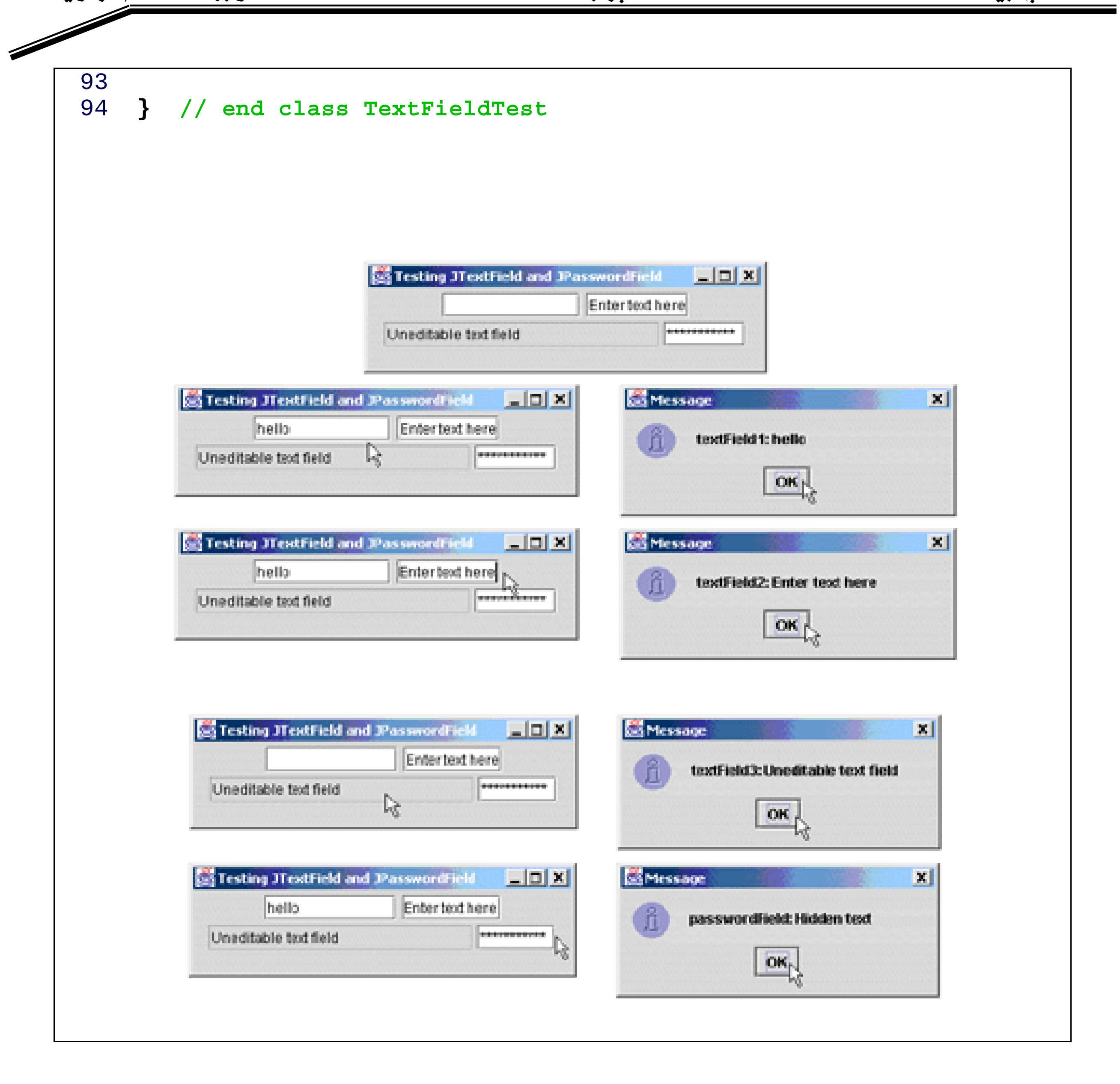
البرنامج التالي يوضح استخدام العنصرين الرسوميين JTextField و JPasswordField والطرق الخاصه بهما:

```
1
2    // Demonstrating the JTextField class.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   public class TextFieldTest extends JFrame {
12      private JTextField textField1, textField2, textField3;
13      private JPasswordField passwordField;
14
15      // set up GUI
16      public TextFieldTest()
17      {
18       super( "Testing JTextField and JPasswordField" );
19
20       Container container = getContentPane();
21       container.setLayout( new FlowLayout() );
22
23       // construct textfield with default sizing
24       textField1 = new JTextField( 10 );
25       container.add( textField1 );
26
27       // construct textfield with default text
28       textField2 = new JTextField( "Enter text here" );
29       container.add( textField2 );
30
31       // construct textfield with default text and
32       // 20 visible elements and no event handler
33     textField3 = new JTextField( "Uneditable text field", 20 );
34       textField3.setEditable( false );
35       container.add( textField3 );
36
37       // construct textfield with default text
38       passwordField = new JPasswordField( "Hidden text" );
```

```
39          container.add( passwordField );
40
41          // register event handlers
42          TextFieldHandler handler = new TextFieldHandler();

43          textField1.addActionListener( handler );
44          textField2.addActionListener( handler );
45          textField3.addActionListener( handler );
46          passwordField.addActionListener( handler );
47
48            setSize( 325, 100 );
49         setVisible( true );
50      }
51
52      // execute application
53      public static void main( String args[] )
54      {
55         TextFieldTest application = new TextFieldTest();
56
57         application.setDefaultCloseOperation(
58            JFrame.EXIT_ON_CLOSE );
59      }
60
61      // private inner class for event handling
62      private class TextFieldHandler implements ActionListener {
63
64         // process text field events
65         public void actionPerformed( ActionEvent event )
66         {
67            String string = "";
68
69            // user pressed Enter in JTextField textField1
70            if ( event.getSource() == textField1 )
71               string = "textField1: " + event.getActionCommand();
72
73            // user pressed Enter in JTextField textField2
74            else if ( event.getSource() == textField2 )
75               string = "textField2: " + event.getActionCommand();
76
77            // user pressed Enter in JTextField textField3
78            else if ( event.getSource() == textField3 )
79               string = "textField3: " + event.getActionCommand();
80
81            // user pressed Enter in JTextField passwordField
82            else if ( event.getSource() == passwordField ) {
83               JPasswordField pwd =
84                  ( JPasswordField ) event.getSource();
85               string = "passwordField: " +
86                   new String( passwordField.getPassword() );
87            }
88
89            JOptionPane.showMessageDialog( null, string );
90         }
91
92      }  // end private inner class TextFieldHandler
```

```
93
94  }  // end class TextFieldTest
```



السطر١٢ -١٣ يعرف ثلاثة عناصر من نوع JTextField و عنصر من نوع JPasswordField، وكلا من هذه العناصر قد تم انشاؤه من خلال البيانات constructors في الأسطر ١٦ -٥٠. السطر ٢٤ يعرف الحقل النصي textField1 بطول ١٠ خانات. السطر ٢٥ يضيف الحقل النصي textField1 إلى شريط المحتويات content pane.

السطر ٢٨ يعرف حقل نصي آخر هو textField2 مع نص أولي "Enter Text Here" ليظهر في الحقل النصي. لاحظ أننا لم نحدد طول الحقل النصي حيث إن طول هذا الحقل سيكون مساوياً لطول النص بداخله. السطر ٢٩ يضيف الحقل النصي إلى شريط المحتويات content pane.

السطر ٣٣ يعرف حقل نصي آخر هو textField3 وينادي باني الفصيلة JTextField مع باراميترين هما النص الأولي "Uneditable Text field" ,وعدد خانات(٢٠) يمثل طول الحقل النصي. السطر ٣٤ يستخدم الطريقة setEditable والباراميتر false ، لتحديد أن المستخدم لا يمكنه تعديل محتويات الحقل. السطر ٣٥ يضيف الحقل النصي إلى شريط المحتويات content pane.

السطر ٣٨ يعرف عنصر passwordField من نوع PasswordField مع النص "Hidden Text" ليظهر في داخل الحقل، وطول هذا العنصر يحدد من خلال طول النص المحدد داخل الحقل. لاحظ أن النص المدخل سيظهر على شكل مجموعة من النجوم asterisks كنوع من الأمان. السطر ٣٩ يضيف حقل كلمة المرور إلى شريط المحتويات content pane.

ولمعالجة الأحداث في هذا البرنامج قمنا بإنشاء صنف خاص TextFieldHandler داخل الفصيلة الرئيسة (السطر ٦٢ -٩٢) والذي يرث الواجهة interface المسماة ActionListener وبالتالي فإن كل عنصر من نوع TextFieldHandler هو عنصر من نوع ActionListener . السطر ٤٢ يعرف عنصر مرجعي من فصيلة TextFieldHandler اسمه Handler، والذي سيستخدم كمستمع للأحداث event-listener على عناصر الحقول النصية وعنصر حقل كلمة المرور.

الأسطر ٤٣ -٤٦ تحتوي على جمل تسجيل عناصر الحقول النصية وكلمة المرور في المستمع handler بعد تنفيذ هذه الجمل، فإن أي حدث (الضغط على زر enter) على العناصر المسجلة في المستمع سيؤدي إلى إطلاق الطريقة actionPerformed.

تقوم الطريقة actionPerformed بتحديد العنصر الذي تسبب في إطلاقها وذلك باستخدام الطريقة getSource الموجودة داخل الفصيلة ActionEvent، وبعد تحديد العنصر نقوم ببناء جملة نصية وتخزينها في متغير اسمه string من نوع String استخدمنا فيها الطريقة getActionCommand الموجودة في الصنف ActionEvent والتي تستخدم لاسترجاع النص من عنصر حرفي TextField والطريقة getPassword الخاصة بالصنف JPasswordField لاسترجاع النص من عنصر كلمة المرور.

العنصر الرسومي (زر) JButton

الزر هو عنصر رسومي يضغطه المستخدم لإطلاق عمل معين، يمكنك عند كتابة برنامج جافا استخدام ازرار من عدة أنواع مثل:

command buttons, check boxes, radio buttons، وسنتطرق للأنواع الثلاثة في الصفحات التالية ونعرض مثالاً على كل منها.

المثال التالي يوضح استخدام زر الأمر Command Buttons. هذا النوع من الأزرار يسبب أطلاق حدث من نوع ActionEvent عند الضغط عليه بالفأرة من قبل المستخدم. ويتم إنشاء عنصر زر الأمر من خلال الصنف JButton.

```
1
2     // Creating JButtons.
3
4     // Java core packages
5     import java.awt.*;
6     import java.awt.event.*;
7
8     // Java extension packages
9     import javax.swing.*;
10
11    public class ButtonTest extends JFrame {
12       private JButton plainButton, fancyButton;
13
14       // set up GUI
15       public ButtonTest()
16       {
17          super( "Testing Buttons" );
18
19          // get content pane and set its layout
20          Container container = getContentPane();
21          container.setLayout( new FlowLayout() );
22
23          // create buttons
24          plainButton = new JButton( "Plain Button" );
25          container.add( plainButton );
26
27          Icon bug1 = new ImageIcon( "bug1.gif" );
28          Icon bug2 = new ImageIcon( "bug2.gif" );
29          fancyButton = new JButton( "Fancy Button", bug1 );
30          fancyButton.setRolloverIcon( bug2 );
31          container.add( fancyButton );
32
33          // create an instance of inner class ButtonHandler
34          // to use for button event handling
35          ButtonHandler handler = new ButtonHandler();
36          fancyButton.addActionListener( handler );
37          plainButton.addActionListener( handler );
```

```
38
39          setSize( 275, 100 );

40          setVisible( true );
41       }
42
43       // execute application
44       public static void main( String args[] )
45       {
46          ButtonTest application = new ButtonTest();
47
48          application.setDefaultCloseOperation(
49             JFrame.EXIT_ON_CLOSE );
50       }
51
52       // inner class for button event handling
53       private class ButtonHandler implements ActionListener {
54
55          // handle button event
56          public void actionPerformed( ActionEvent event )
57          {
58             JOptionPane.showMessageDialog( null,
59                "You pressed: " + event.getActionCommand() );
60          }
61
62       }  // end private inner class ButtonHandler
63
64    }  // end class ButtonTest
```

Testing Buttons
Plain Button | Fancy Button

Testing Buttons
Plain Button | Fancy Button

Message
You pressed: Plain Button
OK

Testing Buttons
Plain Button | Fancy Button

Testing Buttons
Plain Button | Fancy Button

Message
You pressed: Fancy Button
OK

المثال أعلاه يقوم بانشاء عنصرين من نوع JButtons، العناصر من هذا النوع يمكنها إظهار الأيقونات إلى جانب إظهارها للنص كما هو الحال في العناصر من نوع JLabel. معالجة الأحداث للأزرار تتم من خلال كائن من نوع الصنف الداخلي inner class المسمى ButtonHandler (الأسطر ٥٣ -٦٢).

السطر ١٢ يعرف العنصرين من نوع أزرار الأوامر JButton هما: plainButton و fancyButton واللذان تم اعطاؤهما القيم المبدئية في باني الصنف.

السطر ٢٤ ينشئ plainButton ويعطيه النص "plain button" كعنوان لهذا الزر. السطر ٢٥ يضيف العنصر على شريط المحتويات content pane.

يستطيع صنف الأزرار JButton إظهار الأيقونات على الأزرار لتحسين شكل الواجهة الرسومية، كما أن هذا الصنف يوفر خاصية أيقونة المرور على الزر rollover icon وهي الأيقونة التي ستظهر وتختفي على الزر عند مرور الفأرة فوق الزر وخارجه. السطر ٢٧ -٢٨ ينشئ عنصرين من نو ع ImageIcon يمثلان أيقونة الزر الرئيسة وأيقونة المرور على الزر rolover icon للزر المنشأ في السطر ٢٩. كلا السطرين يفترضان أن ملفات الصور مخزنة في نفس الدليل المخزن فيه برنامج جافا.

السطر ٢٩ ينشئ الزر fancyButton مع نص مبدئي هو " Fancy Button " والأيقونة bug1. الوضع التلقائي أن يكون النص على يمين الأيقونة. السطر ٣٠ يستخدم الطريقة setRolloverIcon لتحديد الأيقونة التي ستظهر على الزر عند وضع الفأرة على الزر. السطر ٣١ يضيف الزر إلى شريط المحتويات content pane.

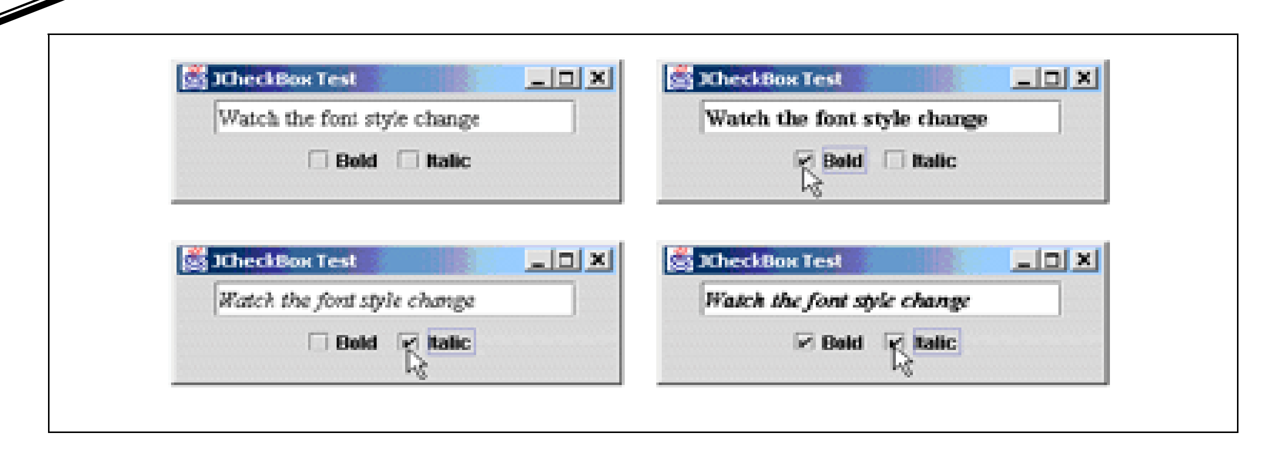
إن الأزرار JButton (مثل الحقول النصيه JTextField) يولد ActionEvent، السطر ٣٥ -٣٧ يسجل كائن مستمع listener لكل زر في البرنامج. السطر ٥٣ -٦٢ يعرف صنف داخلي ButtonHandler والذي يحتوي على الطريقة actionPerformed، تقوم هذه الطريقة بإظهار صندوق رسالة يحتوي على النص الموجود داخل الزر الذي تم معالجته من قبل المستخدم.

العنصر الرسومي JCheckBox.

المثال التالي يوضح كيفية تعريف واستخدام العنصر الرسومي JCheckBox، يقوم هذا البرنامج بتعريف عنصرين من نوع JCheckBox لتغير شكل الخط (أسود، مائل) المكتوب في حقل نصي JTextField فإذا تم اختيار صندوق شكل الخط أسود سيتحول الخط إلى الأسود، وإذا تم اختيار صندوق مائل سيتحول النص إلى مائل، وإذا تم اختيار الصندوقين سيتحول النص إلى أسود ومائل. عند بداية تنفيذ البرنامج لن يكون أي من الصندوقين في وضع الاختيار.

```
1
2    // Creating Checkbox buttons.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   public class CheckBoxTest extends JFrame {
12      private JTextField field;
13      private JCheckBox bold, italic;
14
15      // set up GUI
16      public CheckBoxTest()
17      {
18         super( "JCheckBox Test" );
19
20         // get content pane and set its layout
21         Container container = getContentPane();
22         container.setLayout( new FlowLayout() );
23
24         // set up JTextField and set its font
25         field =
26            new JTextField( "Watch the font style change", 20 );
27         field.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
28         container.add( field );
29
30         // create checkbox objects
31         bold = new JCheckBox( "Bold" );
32         container.add( bold );
33
34         italic = new JCheckBox( "Italic" );
35         container.add( italic );
36
37         // register listeners for JCheckBoxes
38         CheckBoxHandler handler = new CheckBoxHandler();
39         bold.addItemListener( handler );
40         italic.addItemListener( handler );
41
42         setSize( 275, 100 );
43         setVisible( true );
```

```
44      }
45
46      // execute application
47      public static void main( String args[] )
48      {
49         CheckBoxTest application = new CheckBoxTest();
50
51         application.setDefaultCloseOperation(
52            JFrame.EXIT_ON_CLOSE );
53      }
54
55      // private inner class for ItemListener event handling
56      private class CheckBoxHandler implements ItemListener {
57         private int valBold = Font.PLAIN;
58         private int valItalic = Font.PLAIN;
59
60         // respond to checkbox events
61         public void itemStateChanged( ItemEvent event )
62         {
63            // process bold checkbox events
64            if ( event.getSource() == bold )
65
66               if ( event.getStateChange() == ItemEvent.SELECTED )
67                  valBold = Font.BOLD;
68               else
69                  valBold = Font.PLAIN;
70
71            // process italic checkbox events
72            if ( event.getSource() == italic )
73
74               if ( event.getStateChange() == ItemEvent.SELECTED )
75                  valItalic = Font.ITALIC;
76               else
77                  valItalic = Font.PLAIN;
78
79            // set text field font
80            field.setFont(
81               new Font( "Serif", valBold + valItalic, 14 ) );
82         }
83
84      }  // end private inner class CheckBoxHandler
85
86   }  // end class CheckBoxTest
```



بعد إنشاء الحقل النصي وتحديد النص المبدأي بداخله، السطر ٢٧ قام بتحديد التنسيق للخط ليكون نوعه serif، وشكله عادي PLAIN و حجمه ١٤ نقطة. ثم يقوم باني الفصيل بإنشاء عنصرين من نوع JCheckBox في الأسطر ٣١ -٣٤. إن النص الحرفي المرسل كباراميتر لباني الفصيل يمثل النص الذي سيظهر على يمين الحقل JCheckBox.

عندما يقوم المستخدم بالضغط على أحد العنصرين، يؤدي ذلك إلى أطلاق حدث من نوع ItemEvent والذي يمكن معالجته من خلال المستمع ItemListener، ويتوجب على هذا المستمع تعريف الطريقة itemStateChanged لبرمجة المعالجة الخاصة بهذا العنصر.

يتم مناداة الطريقة itemStateChanged في حالة قام المستخدم بالضغط على أحد العنصرين bold أو italic. حيث تستخدم الطريقة event.getSource() لتحديد أي العنصرين تم الضغط عليه. ففي حال أنه العنصر bold فإن جملة if/else في الأسطر ٦٦ -٦٩ تستخدم الطريقة getStateChanged المعرفة في الفصيل ItemEvent لتحديد حالة العنصر: هل هو في حالة اختيار أم عدم اختيار؟ (ItemEvent.SELECTED أو ItemEvent.DESELECTED) اذا كانت الحالة اختيار فإن القيمة العددية للثابت Font.BOLD يتم إسنادها للمتغير العددي valBold وإلا فإن القيمة العددية للثابت Font.PLAIN تسند له. نفس جملة الشرط تعاد للعنصر italic بحيث إنه إذا كانت حالة العنصر اختيار فإن القيمة العددية للثابت Font.ITALIC تسند للمتغير العددي valItalic والا فإن القيمة العددية للثابت Font.PLAIN تسند له. مجموع الحالتين valBold و valItalic استخدمت في الأسطر ٨٠ -٨١ كشكل للخط في الحقل النصي JTextField.

العنصر الرسومي JRadioButton.

تتشابه العناصر الرسومية من نوع JRadioButton مع العناصر من نوع JCheckBox في كون كل منها له حالتان مختار، وغير مختار (selected and deselected) إلا أن radio buttons تظهر غالبا على شكل مجموعة بحيث إن أحد عناصرها يتم اختياره فقط والباقي غير مختار. فعند الضغط على خيار آخر في المجموعة فإن الخيار الأول يتم إلغاؤه deselected. ولجمع عدد من JRadioButtons في مجموعة واحدة سوف نستخدم كائناً من نوع ButtonGroup والذي لا يعتبر عنصر رسومي(على الرغم من وجوده في الحزمة javax.swing) فهو لا يظهر على الشاشة، ووظيفته تنحصر في تحديد العناصر من نوع JRadioButtons التي تمثل مجموعة واحدة.

المثال التالي شبيه بالمثال الخاص بالعنصر الرسومي JCheckBox أعلاه، حيث يستطيع المستخدم تغيير تنسيق الخط في الحقل النصي. يستخدم هذا البرنامج radio buttons لتطبيق تنسيق واحد فقط على النص.

```
1
2     // Creating radio buttons using ButtonGroup and JRadioButton.
3
4     // Java core packages
5     import java.awt.*;
6     import java.awt.event.*;
7
8     // Java extension packages
9     import javax.swing.*;
10
11    public class RadioButtonTest extends JFrame {
12     private JTextField field;
13     private Font plainFont, boldFont, italicFont, boldItalicFont;
14     private JRadioButton plainButton, boldButton, italicButton,
15             boldItalicButton;
16     private ButtonGroup radioGroup;
17
18       // create GUI and fonts
19       public RadioButtonTest()
20       {
21          super( "RadioButton Test" );
22
23          // get content pane and set its layout
24          Container container = getContentPane();
25          container.setLayout( new FlowLayout() );
26
27          // set up JTextField
28          field =
29             new JTextField( "Watch the font style change", 25 );
30          container.add( field );
31
```

```
32          // create radio buttons
33          plainButton = new JRadioButton( "Plain", true );
34          container.add( plainButton );
35
36          boldButton = new JRadioButton( "Bold", false);
37          container.add( boldButton );
38
39          italicButton = new JRadioButton( "Italic", false );
40          container.add( italicButton );
41
42          boldItalicButton = new JRadioButton(
43             "Bold/Italic", false );
44          container.add( boldItalicButton );
45
46          // register events for JRadioButtons
47          RadioButtonHandler handler = new RadioButtonHandler();
48          plainButton.addItemListener( handler );
49          boldButton.addItemListener( handler );
50          italicButton.addItemListener( handler );
51          boldItalicButton.addItemListener( handler );
52
53          // create logical relationship between JRadioButtons
54          radioGroup = new ButtonGroup();
55          radioGroup.add( plainButton );
56          radioGroup.add( boldButton );
57          radioGroup.add( italicButton );
58          radioGroup.add( boldItalicButton );
59
60          // create font objects
61          plainFont = new Font( "Serif", Font.PLAIN, 14 );
62          boldFont = new Font( "Serif", Font.BOLD, 14 );
63          italicFont = new Font( "Serif", Font.ITALIC, 14 );
64          boldItalicFont =
65             new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
66          field.setFont( plainFont );
67
68          setSize( 300, 100 );
69          setVisible( true );
70       }
71
72       // execute application
73       public static void main( String args[] )
74       {
75          RadioButtonTest application = new RadioButtonTest();
76
77          application.setDefaultCloseOperation(
78             JFrame.EXIT_ON_CLOSE );
79       }
80
81       // private inner class to handle radio button events
82       private class RadioButtonHandler implements ItemListener {
83
84          // handle radio button events
85          public void itemStateChanged( ItemEvent event )
86          {
```

```
87              // user clicked plainButton
88              if ( event.getSource() == plainButton )
89                 field.setFont( plainFont );
90
91              // user clicked boldButton
92              else if ( event.getSource() == boldButton )
93                 field.setFont( boldFont );
94
95              // user clicked italicButton
96              else if ( event.getSource() == italicButton )
97                 field.setFont( italicFont );
98
99              // user clicked boldItalicButton
100             else if ( event.getSource() == boldItalicButton )
101                field.setFont( boldItalicFont );
102          }
103
104       }  // end private inner class RadioButtonHandler
105
106    }  // end class RadioButtonTest
```

السطر ٣٣ -٤٤ يعرف كل عنصر من عناصر JRadioButton ويضيفه إلى شريط المحتويات content pane كل كائن من هذه الكائنات تم إنشاؤه وإعطاؤه قيماً من خلال بانيات الفصيل كما في السطر ٣٣، هذا الباني يزود كل عنصر من JRadioButton بعنوان (label) يظهر إلى يمين العنصر، و حالة العنصر. حيث إن القيمه true تعني أن هذا العنصر يجب أن يظهر في الاختيار select. عناصر JRadioButtons مثل عناصر JCheckBox تطلق حدثاً من نوع ItemEvent عندما يتم الضغط عليها. الأسطر ٤٧ -٥١ ينشئ كائن من الفصيل الداخلي RadioButtonHandler (والمعرف في الأسطر ٨٢ -١٠٤) وتسجيله لمعالجة الأحداث ItemEvent التي ستنطلق عند ضغط المستخدم على أي من عناصر JRadioButtons.

السطر ٥٤ يعرف كائناً من نوع ButtonGroup اسمه radioGroup، سيستخدم هذا الكائن لربط العناصر من نوع JRadioButton في مجموعة واحدة بحيث يتم اختيار واحد فقط من هذه العناصر في الوقت الواحد. الأسطر ٥٥ -٥٨ تستخدم الطريقة add المعرفة داخل الفصيل ButtonGroup لربط كل عنصر من نوع JRadioGroup بالمجموعة المسماة radioGroup.

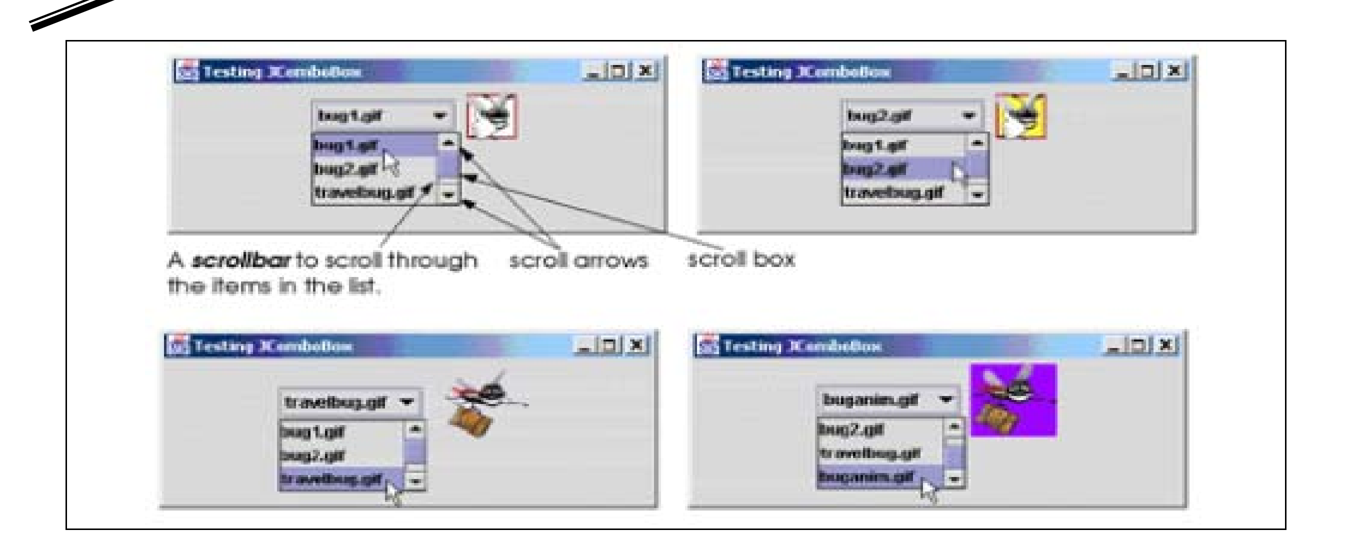
الفصيل RadioButtonHandler (الأسطر ٨٢ -١٠٤) يطبق الواجهة ItemListener وبالتالي فإنه يمكنه معالجة الأحداث من نوع ItemEvent الناتجة عن عناصر JRadioButton. عند الضغط على أي عنصر JRadioButton فإن المجموعة radioGroup تلغي اختيار العنصر السابق وتختار العنصر الحالي وتنفذ الطريقة itemStateChanged (الأسطر ٨٥ -١٠٢) حيث تقوم بتحديد العنصر الذي تم الضغط عليه باستخدام الطريقة getSource، ثم تغير تنسيق الحقل النصي إلى التنسيق الجديد.

العنصر الرسومي JComboBox

عنصر القوائم JComboBox يوفر إمكانية عمل قائمة من الخيارات يستطيع المستخدم الاختيار منها. عنصر القوائم JComboBox مثله مثل العنصرين JCheckBox و JRadioButton يتسبب في إطلاق الحدث ItemEvent عند الضغط عليه. المثال التالي يوضح كيفية تعريف واستخدام عنصر القوائم.

```
1
2    // Using a JComboBox to select an image to display.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   public class ComboBoxTest extends JFrame {
12      private JComboBox imagesComboBox;
13      private JLabel label;
14
15      private String names[] =
16         { "bug1.gif", "bug2.gif",  "travelbug.gif", "buganim.gif" };
17      private Icon icons[] = { new ImageIcon( names[ 0 ] ),
18         new ImageIcon( names[ 1 ] ), new ImageIcon( names[ 2 ] ),
19         new ImageIcon( names[ 3 ] ) };
20
21      // set up GUI
22      public ComboBoxTest()
23      {
24         super( "Testing JComboBox" );
```

```
25
26          // get content pane and set its layout
27          Container container = getContentPane();
28          container.setLayout( new FlowLayout() );
29
30          // set up JComboBox and register its event handler
31          imagesComboBox = new JComboBox( names );
32          imagesComboBox.setMaximumRowCount( 3 );
33
34          imagesComboBox.addItemListener(
35
36             // anonymous inner class to handle JComboBox events
37             new ItemListener() {
38
39                // handle JComboBox event
40                public void itemStateChanged( ItemEvent event )
41                {
42                   // determine whether check box selected
43                   if ( event.getStateChange() == ItemEvent.SELECTED )
44                      label.setIcon( icons[
45                         imagesComboBox.getSelectedIndex() ] );
46                }
47
48             }  // end anonymous inner class
49
50          ); // end call to addItemListener
51
52          container.add( imagesComboBox );
53
54          // set up JLabel to display ImageIcons
55          label = new JLabel( icons[ 0 ] );
56          container.add( label );
57
58          setSize( 350, 100 );
59          setVisible( true );
60       }
61
62       // execute application
63       public static void main( String args[] )
64       {
65          ComboBoxTest application = new ComboBoxTest();
66
67          application.setDefaultCloseOperation(
68             JFrame.EXIT_ON_CLOSE );
69       }
70
71    }  // end class ComboBoxTest
```



يستخدم هذا المثال عنصر القوائم JComboBox لتوفير قائمة من أربع خيارات تمثل أسماء ملفات من نوع صور، وعند الضغط على خيار ما في القائمة، ستظهر الصورة الموجودة في الملف على شكل أيقونة Icon داخل عنصر رسومي من نوع JLabel .

السطر ١٧ -١٩ يعرف مصفوفة اسمها icons ويعطيها القيم الأولية، تحتوي المصفوفة على اربع كائنات من نوع ImageIcon، كما يعرف مصفوفه أخرى من نوع String اسمها names تحتوي على أسماء ملفات الصور المخزنة في نفس الدليل الذي يحتوي على البرنامج.

السطر ٣١ ينشئ كائناً من نوع قائمة JComboBox ويستخدم عناصر المصفوفة names كعناصر للقائمه. مؤشر عددي يتابع ترتيب العناصر في القائمة. العنصر الأول يضاف في الموقع (٠) في القائمة، العنصر الثاني يضاف في الموقع (١) من القائمة وهكذا. العنصر الأول في القائمة يظهر في وضعية الاختيار عند إظهار القائمة على الشاشة. باقي العناصر يتم اختيارها بالضغط عليها من القائمة.

السطر ٣٢ يستخدم الطريقة setMaximumRowCount المعرفة في الفصيلة JComboBox لتحديد الحد الأقصى من العناصر التي ستظهر عند الضغط على القائمه. وفي حالة وجود عدد من العناصر أكبر من الحد الأقصى الممكن إظهاره فإنه يظهر في القائمه شريط تصفح عمودي لتمكين المستخدم من إظهار العناصر المتبقية. السطر ٣٤ -٥٠ يسجل كائناً من نوع الفصيل الداخلي (بدون اسم) والذي يطبق الواجهة ItemListener، حيث تم تسجيله كمستمع على القائمه imagesComboBox. فعندما يختار المستخدم أحد العناصر من القائمه فإن الطريقة itemStateChanged تنفذ (الأسطر ٤٠ -٤٨)

وتقوم بوضع الأيقونة لحقل label. يتم اختيار الأيقونة من مصفوفة icons بعد تحديد موقع العنصر بالمصفوفة بواسطة الطريقة getSelectedIndex في السطر ٤٥.

مديرو عرض العناصر الرسومية Layout Managers

تم تزويد العناصر الرسومية GUI Components الموجودة ضمن حاويات Containers بمديري عرض العناصر الرسومية وذلك لأغراض عرض العناصر في واجهة التطبيق داخل الحاوية Container بشكل منسق. الشكل التالي يوضح ثلاثة أنواع من مديري عرض العناصر الرسومية:

مدير العرض	الوصف
FlowLayout	هذا العرض هو العرض التلقائي لكل من java.awt.Applet, java.awt.Panel, javax.swing.Jpanel. ويقوم بعرض العناصر الرسومية بشكل متسلسل حسب ترتيب إدراجها في حاوية العناصر.
BorderLayout	يستخدم كعرض تلقائي لشريط المحتويات Content pane الخاص بـ JFrame و JApplet، يقوم بترتيب العناصر في خمس مناطق: الشمالية، الجنوبية، الشرقية، الغربية والوسطى.
GridLayout	يقوم هذا العارض بترتيب العناصر في سطور وأعمده.

واليك الأمثلة التالية لتوضيح كل من الأنواع الثلاثة أعلاه.

مدير العرض FlowLayout

```
1
2    // Demonstrating FlowLayout alignments.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   public class FlowLayoutDemo extends JFrame {
12      private JButton leftButton, centerButton, rightButton;
13      private Container container;
14      private FlowLayout layout;
```

```
15
16        // set up GUI and register button listeners
17        public FlowLayoutDemo()
18        {
19           super( "FlowLayout Demo" );
20
21           layout = new FlowLayout();
22
23           // get content pane and set its layout

24           container = getContentPane();
25           container.setLayout( layout );
26
27           // set up leftButton and register listener
28           leftButton = new JButton( "Left" );
29
30           leftButton.addActionListener(
31
32              // anonymous inner class
33              new ActionListener() {
34
35                 // process leftButton event
36                 public void actionPerformed( ActionEvent event )
37                 {
38                    layout.setAlignment( FlowLayout.LEFT );
39
40                    // re-align attached components
41                    layout.layoutContainer( container );
42                 }
43
44              }  // end anonymous inner class
45
46           ); // end call to addActionListener
47
48           container.add( leftButton );
49
50           // set up centerButton and register listener
51           centerButton = new JButton( "Center" );
52
53           centerButton.addActionListener(
54
55              // anonymous inner class
56              new ActionListener() {
57
58                 // process centerButton event
59                 public void actionPerformed( ActionEvent event )
60                 {
61                    layout.setAlignment( FlowLayout.CENTER );
62
63                    // re-align attached components
64                    layout.layoutContainer( container );
65                 }
66              }
67           );
68
69           container.add( centerButton );
```

```
70
71          // set up rightButton and register listener
72          rightButton = new JButton( "Right" );
73
74          rightButton.addActionListener(
75
76             // anonymous inner class
77             new ActionListener() {
78
79                // process rightButton event
80                public void actionPerformed( ActionEvent event )
81                {

82                   layout.setAlignment( FlowLayout.RIGHT );
83
84                   // re-align attached components
85                   layout.layoutContainer( container );
86                }
87             }
88          );
89
90          container.add( rightButton );
91
92          setSize( 300, 75 );
93          setVisible( true );
94       }
95
96       // execute application
97       public static void main( String args[] )
98       {
99          FlowLayoutDemo application = new FlowLayoutDemo();
100
101         application.setDefaultCloseOperation(
102            JFrame.EXIT_ON_CLOSE );
103      }
104
105   }  // end class FlowLayoutDemo
```

البرنامج أعلاه يرسم ثلاثة ازرار من نوع JButtons ويضيفهم إلى التطبيق باستخدام FlowLayout يتم وضع العناصر في الوسط تلقائيا، وعند الضغط على زرLeft يتحول وضع العناصر لتبدأ من اليسار وعند الضغط على زر Right يتحول وضع العناصر لتبدأ من اليمين، وكذلك عند الضغط على زر Center يتم توسيط العناصر. لاحظ إنه عند تصغير عرض النافذة فإن الزر الثالث لا يعود له مكان على نفس السطر لذلك سينتقل إلى سطر جديد.

كما تلاحظ في السطر ٢٥ فإنه يتم تحديد Layout للحاوية Container من خلال الطريقة setLayout، كما تلاحظ إنه يمكنك تغيير وضع العناصر الرسومية ابتداء من اليمين أو اليسار أو الوسط، من خلال الطريقة Layout.setAlignment.

مدير العرض BorderLayout

يقوم هذا العارض بتقسيم الحأويه Container إلى خمس مناطق هي: شمالية، جنوبية، شرقية، ووسطى. يمكنك إضافة عنصر واحد لكل من هذه المناطق الخمس. هذا العنصر يمكن أن يكون حأوية container يحتوي على العديد من العناصر بداخله. المنطقة الشمالية والجنوبية تمتد أفقياً حتى نهاية أطراف الحاوية. اما المنطقة الشرقية والغربية فتمتد عموديا بين المنطقتين الشمالية والجنوبية. أما المنطقة المتبقية فهي للمنطقة الوسطى. في حال عدم وجود المنطقة الشمالية والجنوبية فإن كلا من المناطق الشرقية، الوسطى، والغربية تتمدد لتغطية المنطقه الفارغة. وفي حالة عدم وجود المناطق الشرقية والغربية فإن المنطقة الوسطى تتمدد لتغطية المنطقة الفارغة.المثال التالي يوضح استخدام العارض BorderLayout :

```
1
2    // Demonstrating BorderLayout.
3
4    // Java core packages
5    import java.awt.*;
6    import java.awt.event.*;
7
8    // Java extension packages
9    import javax.swing.*;
10
11   public class BorderLayoutDemo extends JFrame
12      implements ActionListener {
13
14      private JButton buttons[];
15      private String names[] = { "Hide North", "Hide South",
16         "Hide East", "Hide West", "Hide Center" };
```

```
17       private BorderLayout layout;
18
19       // set up GUI and event handling
20       public BorderLayoutDemo()
21       {
22          super( "BorderLayout Demo" );
23
24          layout = new BorderLayout( 5, 5 );
25
26          // get content pane and set its layout
27          Container container = getContentPane();
28          container.setLayout( layout );
29
30          // instantiate button objects
31          buttons = new JButton[ names.length ];
32
33          for ( int count = 0; count < names.length; count++ ) {

34             buttons[ count ] = new JButton( names[ count ] );
35             buttons[ count ].addActionListener( this );
36          }
37
38          // place buttons in BorderLayout; order not important
39          container.add( buttons[ 0 ], BorderLayout.NORTH );
40          container.add( buttons[ 1 ], BorderLayout.SOUTH );
41          container.add( buttons[ 2 ], BorderLayout.EAST );
42          container.add( buttons[ 3 ], BorderLayout.WEST );
43          container.add( buttons[ 4 ], BorderLayout.CENTER );
44
45          setSize( 300, 200 );
46          setVisible( true );
47       }
48
49       // handle button events
50       public void actionPerformed( ActionEvent event )
51       {
52          for ( int count = 0; count < buttons.length; count++ )
53
54             if ( event.getSource() == buttons[ count ] )
55                buttons[ count ].setVisible( false );
56             else
57                buttons[ count ].setVisible( true );
58
59          // re-layout the content pane
60          layout.layoutContainer( getContentPane() );
61       }
62
63       // execute application
64       public static void main( String args[] )
```

```
65        {
66            BorderLayoutDemo application = new BorderLayoutDemo();
67
68            application.setDefaultCloseOperation(
69               JFrame.EXIT_ON_CLOSE );
70        }
71
72     }  // end class BorderLayoutDemo
```



مدير العرض GridLayout

يقوم مدير العرض هذا بتقسيم الحاوية container على شكل شبكه Grid بحيث نقوم بوضع العناصر في صفوف وأعمدة، كل خلية في الشبكة لها نفس الطول والعرض، ويتم وضع العناصر في الشبكة ابتداء من الخلية الواقعة أعلى الشبكة من اليسار وتستمر عملية الاضافة من اليسار لليمين حتى يمتلئ الصف ثم ننتقل للصف الذي يليه. المثال التالي يوضح استخدام GridLayout.

```
1
2     // Demonstrating GridLayout.
3
4     // Java core packages
5     import java.awt.*;
6     import java.awt.event.*;
```

```
 7
 8   // Java extension packages
 9   import javax.swing.*;
10
11   public class GridLayoutDemo extends JFrame
12      implements ActionListener {
13
14      private JButton buttons[];
15      private String names[] =
16         { "one", "two", "three", "four", "five", "six" };
17      private boolean toggle = true;
18      private Container container;
19      private GridLayout grid1, grid2;
20
21      // set up GUI
22      public GridLayoutDemo()
23      {
24         super( "GridLayout Demo" );
25
26         // set up layouts
27         grid1 = new GridLayout( 2, 3, 5, 5 );
28         grid2 = new GridLayout( 3, 2 );
29
30         // get content pane and set its layout
31         container = getContentPane();
32         container.setLayout( grid1 );
33
34         // create and add buttons
35         buttons = new JButton[ names.length ];
36
37         for( int count = 0; count < names.length; count++ ) {
38            buttons[ count ] = new JButton( names[ count ] );
39            buttons[ count ].addActionListener( this );
40            container.add( buttons[ count ] );
41         }
42
43         setSize( 300, 150 );
44         setVisible( true );
45      }
46
47      // handle button events by toggling between layouts
48      public void actionPerformed( ActionEvent event )
49      {
50         if ( toggle )
51            container.setLayout( grid2 );
52         else
53            container.setLayout( grid1 );
54
55         toggle = !toggle;   // set toggle to opposite value
56         container.validate();
57      }
58
59      // execute application
60      public static void main( String args[] )
61      {
```

```
62          GridLayoutDemo application = new GridLayoutDemo();
63
64          application.setDefaultCloseOperation(
65             JFrame.EXIT_ON_CLOSE );
66       }
67
68    }  // end class GridLayoutDemo
```

GridLayout Demo		
one	two	three
four	five	six

GridLayout Demo	
one	two
three	four
five	six

تمارين

س١) حدد الأخطاء في كل من الجمل التالية:

```
a) buttonName =JButton("Caption");
b) JLabel aLabel, JLabel;
c) TextField = new JTextField(50, "Default Text");
d) Container c = getContentPane();
   setLayout (new BorderLayout());
   button1 = newJButton ("North Star");
   button2 = newJButton ("South Pole");
   c.add(button1);
   c.add(button2);
```

س٢) قم برسم الشكل التالي من غير اضافة أي عمليات على الرسم

☐	Snap to Grid	X [8]	OK
☐	Show Grid		Cancel
		Y [8]	Help

س٣) قم برسم شكل الآلة الحاسبة التالية، واكتب برنامج جافا ليقوم باجراء العملية التي يختارها المستخدم وطباعة الناتج في صندوق النص أعلى الشكل.

(استخدم العناصر الرسومية JButtons، JTextField في الحل)

7	8	9	/
4	5	6	*
1	2	3	-
0	.	=	+

برمجة ٣

الروتين

الجدارة:

أن يكون المتدرب قادراً على التمييز بين أنواع الملفات المختلفة، وكتابة برنامج جافا و يستطيع التعامل مع الملفات..

الأهـداف:

بنهاية هذه الوحدة، عليك أن تكون قادرا على كتابة برنامج جافا يقوم بما يلي:

١. تعريف كائن يحتوي على الملف المراد معالجته
٢. فتح الملف
٣. القراءة من الملف
٤. الكتابة على الملف
٥. تحديد نهاية الملف

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان الجدارة بنسبة ١٠٠٪

الوسائل المساعدة:

- وجود حاسب آلي
- دفتر
- قلم

معالجة الملفات.

مقدمة

إن تخزين الملفات في متغيرات ومصفوفات هو تخزين مؤقت لها، حيث إن البيانات تفقد عند انتهاء فترة حياة المتغير variable scope أو عند انتهاء تنفيذ البرنامج، لذلك تستخدم البرامج الملفات كوسيلة لتخزين البيانات لفترات زمنية طويلة تتعدى مرحلة تنفيذ البرنامج. يقوم الحاسب بتخزين الملفات في الذاكرة الثانوية مثل القرص الصلب، الأشرطة الممغنطة وغيرها.

في هذه الوحدة سنتعرف على كيفية القراءة من الملفات و الكتابة عليها، ومعالجة البيانات المخزنة على شكل ملفات.

تعتبر عملية معالجة الملفات واحدة من أهم الإمكانيات الواجب توفرها في لغة البرمجة التي ستستخدم لبناء التطبيقات التجارية التي تتعامل مع كمية كبيرة من البيانات، مثل نظام دفع الرواتب.

تنقسم الملفات حسب طريقة الوصول للبيانات فيها إلى:

a. ملفات الوصول التتابعي Sequential Access Files

في هذا النوع من الملفات للوصول إلى سجل معين في الملف عليك قراءة الملف من البداية سجل يليه آخر حتى تصل إلى السجل المطلوب، وكذلك عند الكتابة على الملف

b. ملفات الوصول العشوائي Random Access Files

في هذا النوع من الملفات يمكنك الوصول إلى السجل المطلوب مباشرة دون الحاجة للمرور على كل السجلات التي قبله.

كما ويمكن تقسيم الملفات حسب نوع البيانات إلى:

i. ملفات نصية Text files : حيث تتم قراءة بيانات الملف والكتابة عليه على شكل حروف Characters

ii. ملفات ثنائية Binary files : وهنا يتم التعامل مع بيانات الملف على أنها مجموعة من البايت Bytes

ولكل نوع من الملفات تطبيقات معينة يمكن استخدامه فيها، وستقتصر دراستنا في هذه الوحدة على ملفات الوصول التتابعي والملفات النصية.

القراءة من ملف

إليك الآن المثال الأول، والذي يوضح كيفية تعريف الملفات وفتحها للقراءة منها سطراً سطراً.

```
1     import java.io.*;
2     //Class Definition
3     class ReadTextFilel
4     {
5        public static void main (String args[]) throws IOException
6         {
7             String fileName = "c:/temp/toRead.txt";
8             String line;
9              BufferedReader in = new BufferedReader (new
10                                                    FileReader (filename) );
11             line = in.readLine();
12             while (line != null) // continue until end of file;
13             {
14                 System.out.prinln(line);
15                  Line = in.readLine();
16              }
17             in.close();
18          }
19    }
20
```

السطر الأول: يقوم السطر الأول باستيراد المجموعة package المسماة java.io وتحتوي علىجميع الفصائل الخاصة بالملفات ومعالجتها.

السطر ٢ -٦ : في هذه الأسطر قمنا بتعريف اسم الفصيل ReadTextFile1 والطريقة الرئيسة main، لاحظ أن استخدامنا للملفات قد يتسبب في حدوث بعض الاسثناءات مثل "الملف غير موجود FileNotFound" وغيره لذلك وجب علينا تحديد أن الطريقة الرئيسة main قد تطلق استثناء، وذلك باستخدام الجملة throws IOException.

السطر ٧: لفتح واستخدام ملف معين لابد لنا من تعريف اسم الملف وموقع تخزين الملف وذلك بتحديد المسار الخاص به، وقد قمنا بذلك من خلال تعريف متغير نصي اسمه fileName يمثل اسم الملف والمسار الخاص به.

```
String fileName = "c:/temp/toRead.txt";
```

السطر ٨: كما قمنا بتعريف متغير نصي آخر هو line والذي سنستخدمه لتخزين السطر المقروء من الملف من أجل طباعته على الشاشة.

السطر ٩ -١٠: في هذا السطر نقوم بتعريف كائن اسمه in من نوع BufferedReader حيث إن الكائن سيستخدم في قراءة ملف هو fileName معرف على شكل كائن للقراءة من نوع FileReader.

```
BufferedReader in = new BufferedReader  (new FileReader (fileName) );
```

السطر ١٢: بعد ان قمنا بتعريف الكائن in والذي سنقرأ من خلاله البيانات من الملف على شكل سطر يليه سطر. نقوم الآن بقراءة السطر الأول

```
line = in.readln ();
```

في هذه الجمله نطلب قراءة سطر من الكائن in، ثم تخزين السطر في المتغير line والانتقال إلى السطر التالي.

السطر ١١: للاستمرار في قراءة البيانات من الملف وحيث إن عدد الأسطر فيه غير محدد فإننا سنستخدم التكرار (طالما while) لتحديد متى ينتهي الملف حيث إن الملف ينتهي عندما لا يعود هناك أسطر للقراءة أي عند قراءة null.

```
while (line != null)
```

السطر ١٢ -١٥: نقوم هنا بطباعة السطر المقروء على الشاشة، والانتقال لقراءة السطر الذي يليه، وهكذا حتى ننتهي من قراءة وطباعة الملف كله.

```
{
     System.out.prinln(line);
     line = in.readLine();
}
```

السطر ١٦: بعد فتح الملف والانتهاء من قراءة البيانات منه، نقوم الآن بإغلاق الملف وإنهاء عمل الكائن in.

```
in.close();
```

بعد أن تعرفنا على كيفية قراءة الملف سطر يليه سطر، سنتعرف الآن على كيفية القراءة من الملف ولكن كلمة تليها كلمة، حيث إن هناك العديد من التطبيقات التي نحتاج فيها إلى قراءة كلمات من الملف.

```
1    import java.io.*;
2    class ReadWithTokenizer
3    {
4       public static void main (String args[]) throws IOException
5       {
6
7            String fileName = "c:/temp/toRead.txt";
8            BufferedReader in = new BufferedReader  (
9                                              new FileReader (fileName) );
10            StreamTokenizer reader = new StreamTokenizer(in);
11            reader.nextToken();
12           While (reader.ttype != StreamTokenizer.TT_EOF)
13             //continue until end of file
14             {
15                   String word = reader.sval;
16                   System.out.println (word);
17                   Reader.nextToken();
18             }
19             in.close();
20        }
21   }
22
23
```

السطر الأول: يقوم السطر الأول باستيراد المجموعة package المسماة java.io وتحتوي علىجميع الفصائل الخاصة بالملفات ومعالجتها.

السطر ٢ -٦ : في هذه الأسطر قمنا بتعريف اسم الفصيلة ReadwithTokenizer والطريقة الرئيسة main.

السطر ٧: نقوم هنا بتعريف اسم الملف وموقع تخزينه، وقد قمنا بذلك من خلال تعريف متغير نصي اسمه fileName يمثل اسم الملف والمسار الخاص به.

```
String fileName = "c:/temp/toRead.txt";
```

السطر ٨ -٩: في هذا السطر نقوم بتعريف كائن اسمه in من نوع BufferedReader حيث إن الكائن سيستخدم في قراءة ملف هو fileName معرف على شكل كائن للقراءة من نوع FileReader.

```
BufferedReader in = new BufferedReader (new FileReader (fileName) );
```

السطر ١٠: حيث إننا نريد قراءة الملف على شكل كلمات فإننا سنتعامل مع الملف على أنه مجموعة من الكلمات Tokens لذلك سنقوم بإدخال الكائن الذي يمثل ملف القراءة كباراميتر لمنشئ كائن جديد من نوع StreamTokenizer هو reader، الذي سيمكننا من القراءة بشكل كلمات

```
StreamTokenizer reader = new StreamTokenizer(in);
```

السطر ١١: لقراءة الكلمة الأولى في الملف سنستخدم الطريقة المسماة nextToken() والخاصة بالفصيل StreamTokenizer، وستتم منادات الطريقة من خلال الكائن reader.

```
reader.nextToken();
```

السطر ١٢: سنستمر في قراءة الكلمات من الملف حتى نصل إلى نهايته، حيث إن كل ملف له حرف يمثل نهايته، هذا الحرف مخفي في لغة جافا abstracted، ويمكننا الرجوع له من خلال المعرف الثابت TT_EOF الموجود داخل الفصيل StreamTokenizer، وبناء على ذلك فإن جملة التكرار في البرنامج هي:

```
while (reader.ttype != StreamTokenizer.TT_EOF)
```

السطر ١٥: عند استخدام الطريقة reader.nextToken فإن الكلمة التالية في الملف ستنتقل إلى الكائن reader، هذه الكلمة إما أن يكون لها قيمة حرفيه أو قيمة رقمية لذلك سنقوم بقراءة هذه القيمة وتخزينها في متغير من نوع حرفي أو رقمي باستخدام الطريقة nval أو sval على التوالي.

```
String word = reader.sval;
```

السطر ١٦: سنقوم بطباعة الكلمة من خلال الطريقة القياسية

```
System.out.println (word);
```

السطر ١٧:قبل نهاية التكرار لا بد لنا من قراءة الكلمة التاليه حتى نتمكن من التحقق من الوصول إلى نهاية الملف أو طباعة الكلمة وهكذا.

```
reader.nextToken ();
```

السطر ١٩: بعد الانتهاء من قراءة البيانات من الملف سنقوم باغلاقه من خلال الجملة التالية:

```
in.close ();
```

كما يمكنك التاكد من نوع الكلمة المقروءة من الملف وتخزينها في المتغير المناسب، لاحظ الجمل التالية:

```
While (reader.nextToken() != StreamTokenizer.TT_EOF)
//continue until end of file
{
    if (reader.ttype == StreamTokenizer.TT_WORD)
          System.out.println(“ A word: “ + reader.sval);
    Else if (reader.ttype == StreamTokenizer.TT_NUMBER)
           System.out.println(“ A number: “ + reader.nal);
}
```

لقد استخدمنا في المثال أعلاه مجموعة من المعرفات الثابتة الموجودة داخل الفصيل StreamTokenizer مثل TT_WORD ويمثل هذا المعرف القيم الحرفية String، في حين أن المعرف TT_NUMBER يمثل القيم العددية. كما أن لكل كلمة يتم قراءتها من الملف إلى الكائن reader لها نوع يحدد بالمتغير ttype. وبالتالي نستطيع تكوين جملة شرطية كما يلي:

```
if (reader.ttype == StreamTokenizer.TT_WORD)
```

الكتابة على ملف

لقد رأينا في الأمثلة أعلاه كيف نقرأ من ملف وكيف نتعامل مع البيانات على شكل كلمات أو أسطر، وسوف نتعلم الآن كيف نكتب على ملف. انظر إلى البرنامج التالي:

```
 1  import java.io.*;
 2  public class WriteTextFile
 3  {
 5      public static void main (String args[] throws IOExcepetion
 6      {
 7          String filename = "reaper.txt"
 8           PrintWriter print = new PrintWriter( new BufferedWriter (
 9                                     new FileWriter (filename)));
10           print.println("College of Telecommunication and Information");
11           print.println("Computer Department");
12           print.println("Programming");
13           print.println("Java 3");
14           print.close();
15      }
16  }
```

في السطر ٧ قمنا بتحديد اسم الملف المراد الكتابة عليه، ثم قمنا في السطر الثامن بتعريف كائن اسمه printمن نوع PrintWriter وهو الكائن الذي سنستخدمه للكتابة على الملف.

إن عملية الكتابة على الملف تتم من خلال استخدام الطريقة print أو println مسبوقة باسم الكائن الذي يمثل الملف الخاص بالكتابة.

اخيرا سنعرض لك مثالاً يوضح كيفية مناداة طريقه معرفة من قبل المستخدم لتحميل البيانات من ملف إلى مصفوفة. وطريقه أخرى لتخزين البيانات ونقلها من المصفوفة إلى الملف.

```
1
2    import java.io.*
3    class ReadWrite
4    {
5        public static void main(String[] args) throws IOException
6        {
7            String[] line = new String[10];
8            load (line);
9
10          /* -------- نكتب هنا مجموعة من العمليات المختلفهm ----------------*/
11
12          commit (line);
13      }
14     □ سنعرف الان طريقه لتحميل البيانات من الملف إلى مصفوفة من الكائنات //
15
16     Public static void load (String[] line) throws IOException
17     {
18        String filename =”c:/temp/toRead.txt”;
19        BufferedReader in = new BufferedReader (new
20                                        FileReader(filename));
21        Line[0] = in.readln();
22        int i = 0;
23        while (line[i] != null) // استمر حتى نهاية الملف
24        {
25           System.out.println(line[i]);
26            i++;
27            Line[i] = in.readln();
28        }
29        in.close();
30     }
```

```
31  // سنعرف الآن طريقه لحفظ البيانات التي في المصفوفة إلى الملف
32
33     public static void commit (String[] line) throws IOException
34     {
35         String filename =”c:/temp/toRead.txt”;
36         BufferedWriter print = new BufferedWriter (new
37
38  FileWriter(filename));
39         int i = 0;
40         while (line[i] != null) // استمر حتى نهاية الملف
41         {
42           print.println(line[i]);
43             i++;
44         }
45        print.close();
46     }
47  }
48
```

تمارين

س١) لماذا نحتاج استخدام الملفات خصوصا في البرامج التي تتعامل مع كم كبير من البيانات؟

____

____

____

س٢) تتقسم الملفات حسب طريقة الوصول للبيانات إلى قسمين هما :

١. ________________________________

٢. ________________________________

س٣)) تتقسم الملفات حسب طريقة التعامل مع البيانات إلى قسمين هما :

١. ________________________________

٢. ________________________________

س٤) لديك شركه تحتاج فيها إلى متابعة بيانات الموظفين وبيانات الأقسام وتحديد القسم الذي يعمل فيه كل موظف. قم بكتابة فصيلين تنشئ من خلالهما مجموعة من الأنواع (ADT) لكل من الموظفين والأقسام، ثم اكتب برنامجاً يتعامل مع هذين الفصلين، ويقوم بتخزين البيانات على ملفات وقراءتها من الملفات مرة أخرى.



برمجة ٣

الاتصال بقواعد البيانات

الجدارة:

أن يكون المتدرب قادراً على تعريف مشغل لقاعدة بيانات معينة وكتابة برنامج جافا يقوم بالتعامل مع قاعدة البيانات من خلال هذا المشغل.

الأهـداف:

بنهاية هذه الوحدة، عليك أن تكون قادرا على:

١. إنشاء مشغل بقاعدة البيانات
٢. كتابة برنامج جافا يقوم بالتعامل مع قاعدة البيانات من خلال المشغل

مستوى الأداء المطلوب:

أن يصل المتدرب إلى إتقان الجدارة بنسبة ١٠٠٪

الوسائل المساعدة:

- وجود حاسب آلي
- دفتر
- قلم

اتصال جافا بقواعد البيانات

مقدمه

توفر لغة الجافا كمثيلاتها من لغات البرمجة عدة طرق لتخزين المعلومات على ملفات مخزنة في الذاكرة الثانوية مثل: الملفات التسلسلية Sequential files وملفات الوصول العشوائي Random-access files وعلى الرغم من فائدة هاتين الطريقتين في حفظ البيانات، إلا أنهما لا تمتلكان الإمكانيات الكافية للاستعلام عن البيانات بشكل مناسب. إن أنظمة قواعد البيانات لا توفر فقط القدرة على معالجة الملفات،ولكنها تنظم البيانات بطريقة تسمح لها تنفيذ عمليات استعلام معقده. إن أشهر أنواع قواعد البيانات المستخدمة حاليا هي قواعد البيانات العلائقية Relational Database systems وفي هذا النوع من قواعد البيانات تستخدم لغه تسمى لغة الاستعلام الهيكلية Structured Query Language (SQL) والتي تستخدم للاستعلام عن البيانات التي تحقق شرطاً معيناً في قاعدة البيانات. وللاستفادة من قواعد البيانات هذه فإن معظم لغات البرمجة ومنها جافا توفر إمكانية عمل اتصال بقواعد البيانات المختلفة وكتابة جمل SQL للاستعلام عن البيانات. ومن أشهر قواعد البيانات المستخدمه ما يلي: Microsoft Access, Sybase, Oracle, Informix, Microsoft SQL Server وغيرها. في هذه الوحدة سنتعرف على كيفية كتابة برنامج يتصل بقاعدة بيانات Microsoft Access و Oracle، آخذين بعين الاعتبار إن لدى القارئ الماماً بلغة SQL ومفاهيم قاعدة البيانات.

قاعدة البيانات المستخدمة

سنقوم الآن بوصف قاعدة البيانات المستخدمة في هذه الوحدة واسمها الكتب Books، والتي يمكن انشاؤها كقاعدة بيانات Access أو Oracle أو غيرهما.

تتكون قاعدة البيانات هذه من أربعة جدأول هي: المؤلف، الناشر، ISBN للمؤلف وجدول العنوان، وتفصيلاتها كما يلي:

جدول المؤلف	
الحقل	الوصف
رقم المؤلف AutherID	رقم المؤلف في قاعدة البيانات(المفتاح الرئيس Primary Key)
الاسم الأول FirstName	الاسم الأول للمؤلف
الاسم الأخير LastName	الاسم الأخير للمؤلف
سنة الميلاد YearBorn	السنة التي ولد فيها المؤلف

وإليك عينة من البيانات الموجودة في جدول المؤلف:

AutherId	FirstName	LastName	YearBorn
1	Ali	Suliman	1960
2	Salem	Khalid	1975
3	Abdullah	Amer	1963

جدول الناشر	
الحقل	الوصف
رقم الناشر PublisherID	رقم الناشر في قاعدة البيانات (مفتاح رئيس)
اسم الناشر PublisherName	اسم الناشر

عينة من البيانات الموجودة في جدول الناشر:

PublisherID	PublisherName
1	Prentice Hall
2	Prentice Hall PTR

جدول **ISBN**المؤلف	
الحقل	الوصف
ISBN	رقم ISBN للكتاب
رقم المؤلف AutherID	رقم المؤلف صاحب الكتاب

عينة من البيانات الموجودة في جدول ISBNالمؤلف:

ISBN	AutherID
0-13-010671-2	1
0-13-015231-2	2
0-14-044131-7	1
0-11-028271-4	3
0-10-070471-1	2

جدول العنوان	
الحقل	الوصف
ISBN	رقم ISBN للكتاب
العنوان Title	عنون الكتاب
رقم الطبعة EditionNo	رقم طبعة الكتاب
تاريخ النشر YearPublished	سنة نشر الكتاب
رقم الناشر PublisherId	رقم ناشر الكتاب

عينة من البيانات الموجودة في جدول العنوان:

ISBN	Title	Edition No	Year-Published	PublisherID
0-13-010671-2	C How to program	2	1994	1
0-13-015231-2	C++ How to Program	3	1997	1
0-14-044131-7	Java How to program	2	1992	1
0-11-028271-4	Oracle PL/SQL	2	1999	2
0-10-070471-1	Internet Programming	1	1998	1

الخطوة اللاحقة هي معرفة كيفية تعريف قاعدة البيانات أعلاه بلغة الجافا والتي يمكنك إنشاؤها بأحد أنظمة معالجة قواعد البيانات العلائقية Relational database Management Systems مثل Access ، Oracle أو غيرها.

مثال: استرجع بيانات جدول المؤلفين مرتبة حسب الاسم الأول.

```
SELECT * from Author
ORDER BY FirstName
```

البرنامج الأول:

في هذا المثال سنقوم بإجراء عمليات استعلام بسيطة على قاعدة بيانات الكتب واسترجاع البيانات عن كل المؤلفين وعرضها على وحدة JTable. يوضح البرنامج أدناه كيفية الاتصال بقاعدة البيانات، الاستعلام من قاعدة البيانات وعرض النتائج.

```
1    // Fig. 18.24: TableDisplay.java
2    // This program displays the contents of the Authors table
3    // in the Books database.
4    import java.sql.*;
5    import javax.swing.*;
6    import java.awt.*;
7    import java.awt.event.*;
8    import java.util.*;
9
10   public class TableDisplay extends JFrame {
11     private Connection connection;
12     private JTable table;
13
14     public TableDisplay()
15     {
16       // The URL specifying the Books database to which
17       // this program connects using JDBC to connect to a
18       // Microsoft ODBC database.
19       String url = "jdbc:odbc:Books";
20       String username = "anonymous";
21       String password = "guest";
22
23       // Load the driver to allow connection to the database
24       try {
25         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
26
27         connection = DriverManager.getConnection(
```

```
28                url, username, password );
29           }
30           catch ( ClassNotFoundException cnfex ) {
31
32              System.err.println(
33                 "Failed to load JDBC/ODBC driver." );
34              cnfex.printStackTrace();
35              System.exit( 1 );  // terminate program
36           }
37           catch ( SQLException sqlex ) {
38              System.err.println( "Unable to connect" );
39              sqlex.printStackTrace();
40           }
41
42           getTable();
43
44           setSize( 450, 150 );
45           show();
46        }
47
48        private void getTable()
49        {
50           Statement statement;
51           ResultSet resultSet;
52
53           try {
54              String query = "SELECT * FROM Author";
55
56              statement = connection.createStatement();
57              resultSet = statement.executeQuery( query );
58              displayResultSet( resultSet );
59              statement.close();
60           }
61           catch ( SQLException sqlex ) {
62              sqlex.printStackTrace();
63           }
64        }
65
66        private void displayResultSet( ResultSet rs )
67           throws SQLException
68        {
69           // position to first record
```

```
70          boolean moreRecords = rs.next();
71
72          // If there are no records, display a message
73          if ( ! moreRecords ) {
74            JOptionPane.showMessageDialog( this,
75              "ResultSet contained no records" );
76            setTitle( "No records to display" );
77            return;
78          }
79
80          setTitle( "Authors table from Books" );
81
82          Vector columnHeads = new Vector();
83          Vector rows = new Vector();
84
85          try {
86            // get column heads
87            ResultSetMetaData rsmd = rs.getMetaData();
88
89            for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
90              columnHeads.addElement( rsmd.getColumnName( i ) );
91
92            // get row data
93            do {
94              rows.addElement( getNextRow( rs, rsmd ) );
95            } while ( rs.next() );
96
97            // display table with ResultSet contents
98            table = new JTable( rows, columnHeads );
99            JScrollPane scroller = new JScrollPane( table );
100           getContentPane().add(
101             scroller, BorderLayout.CENTER );
102           validate();
103         }
104         catch ( SQLException sqlex ) {
105           sqlex.printStackTrace();
106         }
107       }
108
109       private Vector getNextRow( ResultSet rs,
110                                  ResultSetMetaData rsmd )
111          throws SQLException
```

```
112        {
113          Vector currentRow = new Vector();
114
115          for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
116            switch( rsmd.getColumnType( i ) ) {
117              case Types.VARCHAR:
118                  currentRow.addElement( rs.getString( i ) );
119                break;
120              case Types.INTEGER:
121                  currentRow.addElement(
122                    new Long( rs.getLong( i ) ) );
123                break;
124              default:
125                System.out.println( "Type was: " +
126                  rsmd.getColumnTypeName( i ) );
127            }
128
129          return currentRow;
130        }
131
132        public void shutDown()
133        {
134          try {
135            connection.close();
136          }
137          catch ( SQLException sqlex ) {
138            System.err.println( "Unable to disconnect" );
139            sqlex.printStackTrace();
140          }
141        }
142
143        public static void main( String args[] )
144        {
145          final TableDisplay app = new TableDisplay();
146
147          app.addWindowListener(
148            new WindowAdapter() {
149              public void windowClosing( WindowEvent e )
150              {
151                app.shutDown();
152                System.exit( 0 );
153              }
```

```
154            }
155         );
156      }
       }
```

لاحظ جملة

import java.sql.*

تقوم هذه الجملة باستيراد الحزمة java.sql والتي تحتوي على كل الفصائل Classes المتعلقة بإدارة قواعد البيانات العلائقية في لغة الجافا. في حين أن جملة

private Connection connection;

تعرف مرجعية اتصال، ويتم تعريف كائن اتصال Connection Object لإدارة الاتصال بين برنامج جافا وبين قاعدة البيانات، كما أنه يوفر إمكانية تنفيذ جملSQL لمعالجة قاعدة البيانات والحركات Transactions الموجه إليها.

إن منشئ الفصيلة Class Constructor للفصيله TableDisplay سيقوم بإنشاء الاتصال مع قاعدة البيانات وعند نجاحه بذلك سينفذ الاستعلام المطلوب ويظهر الناتج من خلال مناداة الدالة getTable.

```
String url = "jdbc:odbc:Books";
String username = "anonymous";
String password = "guest";
```

حتى يتمكن منشئ الفصيلة من إنشاء الاتصال، لا بد من تحديد ثلاثة أمور موضحة في الجمل الثلاث أعلاه وهي: موقع قاعدة البيانات المراد الاتصال بها من خلال تحديد عنوان URL والذي يحدد البروتوكول الرئيس jdbc والبروتوكول الفرعي odbc المستخدمين في تحقيق الاتصال يليهما بيان اسم قاعدة البيانات. كما نحتاج لتحديد اسم المستخدم username وكلمة المرور password التي سيتم من خلالها الاتصال بقاعدة البيانات حيث إننا عند تعريف مصدر البيانات كما سيتضح في الموضوع التالي قد حددنا ضرورة إدخال اسم المستخدم وكلمة المرور .

وليتمكن أي برنامج جافا من الوصول إلى أي قاعدة بيانات بتقنية ODBC فإن لغة جافا توفر مشغل Driver لتعريف كيفية اتمام الاتصال اسمه jdbc.odbc.jdbcodbcDriver ويجب تحميل هذا المشغل قبل إجراء الاتصال مع قاعدة البيانات.

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

إن الجملة أعلاه تستخدم الدالة forName لتحميل تعريف الفصيلة class التي تحتوي على مشغل قاعدة البيانات Database driver، وتحميل هذه الفصيلة قد يتسبب في استثناء في حالة أن الفصيلة المطلوبة لا

يمكن تحميلها وهذا الاستثناء هو java.lang.classNotFound لذلك فقد وضعت في جزء try block لنتمكن من متابعة الاستثناء ومعالجته في جزء catch block.

```
connection = DriverManager.getConnection(url, username, password );
```

في الجملة أعلاه نستخدم الدالة getConnection التابعة للفصيلة DriverManager من أجل الاتصال بقاعدة البيانات المحددة بالمتغير URL وقد تم تحديد اسم المستخدم وكلمة المرور أيضا. وفي حالة عدم القدرة على الاتصال بقاعدة البيانات سيحدث استثناء هو java.sql.sqlException أما في حالة الاتصال بقاعدة البيانات فإنه سيتم مناداة الدالة getTable لاسترجاع البيانات.

تقوم الدالة gettable بالاستعلام من قاعدة البيانات ومن ثم مناداة الدالة displayResultSet لإنشاء كائن رسومي JTable لإظهار الناتج من خلاله.

```
Statement statement
```

السطر أعلاه يعرف مرجع لجملةSQL من نوع Statement (هذا النوع موجود ضمن الحزمة java.sql) وهذا المرجع سيستخدم للرجوع إلى الكائن الذي سيتم حفظ جملة SQL بداخله لنقلها إلى قاعدة البيانات لتنفيذها.

```
ResultSet resultSet
```

في هذا السطر قمنا بتعريف الكائن resultSet والذي سيتم إرجاعه إلى برنامج الجافا من قاعدة البيانات، وبداخله ناتج تنفيذ جملة SQL.

```
statement = connection.createStatement();
```

تقوم الجملة أعلاه بمناداة الدالة CreateStatement لايجاد كائن statement والذي سيستخدم للاستعلام من قاعدة البيانات

```
resultSet = statement.executeQuery(query);
```

تقوم هذه الجملة بعمل الاستعلام من خلال مناداة الدالة executeQuery، هذه الدالة ستعيد كائن من قاعدة البياتات يحتوي على ناتج تنفيذ الاستعلام. الكائن resultSet سيمرر إلى الدالة displayResult وبعدها يتم إغلاق الجملة للدلالة على الانتهاء من معالجة الجملة.

السطر ٦٩ من الدالة displayResultSet

```
boolean moreRecords = rs.next();
```

بعد تنفيذ هذا السطر فإن المؤشر الممثل بالمتغيرmoreRecords يشير إلى السجل الأول في النتائج الموجودة في الكائن ResultSet وذلك باستخدام الدالة next والتي تحرك المؤشر إلى السجل التالي حيث إنه يشير مبدئيا إلى ما قبل السجل الأول. وتلاحظ أن الدالة next ترجع قيمة بووليه Boolean تبين من خلالها فيما إذا كان باستطاعتها الانتقال إلى السجل التالي إن وجد (True)، أو عدم وجود تالي وبالتالي يكون

الناتج (False). في حالة أنه كان هناك تالي فإن السطر ٨١ يعرف مصفوفة Vector لتخزين أسماء الاعمدة الموجودة في الناتج ResultSet والسطر ٨٢ يعرف مصفوفة لتخزين سجلات البيانات من الكائن ResultSet ، هذه المصفوفات ستسخدم مع منشئ JTable لبناء جدول يظهر البيانات من ResultSet.

السطر ٨٦:

```
ResultSetMetaData  rsmd = rs.getMetaData();
```

تقوم هذه الجملة بالحصول على البيانات التفصيلية عن الجدأول الموجودة في ResultSet ،مثل أسماء وأنواع الأعمدة في الجدأول، تسمى هذه البيانات التفصيلية MetaData ومن ثم إسنادها إلى الكائن rsmd. لقد قمنا باستخدام ResultSetMetaData في الأسطر ٨٨ و ٨٩ لاسترجاع اسم كل عمود في الكائن ResultSet وقد تم استخدام الدالة getColumnCount لتحديد عدد الأعمدة والدالة getColumnName لتحديد اسم العمود

السطر ٩٢ لغاية ٩٤:

```
do
      rows.addElement (getNextRow (rs, rsmd ) );
       while (rs.next() );
```

تقوم باسترجاع كل سطر من ResultSet باستخدام الدالة getNextRow ،والمعرفة بالسطر ١٠٨، هذه الدالة لها قيمة مرتجعة من نوع مصفوفه احادية البعد تحتوي البيانات لسطر واحد، لاحظ الشرط rs.next() والذي ينقل المؤشر الخاص بمتابعة الانتقال إلى السجل التالي في الكائن ResultSet، إن وجد، وبالتالي فإن التكرار أعلاه سينتهي عندما لا يبقى هناك سجلات في الكائن ResultSet.

بعد تحويل كل السجلات إلى مصفوفات أحادية البعد، يقوم السطر ٩٧ بإنشاء JTable لإظهار هذه السجلات.

الدالة getNextRow (سطر ١٠٨) تستقبل ResultSet و ResultSetMetaData كباراميترات وتنشئ مصفوفة أحادية تحتوي على سجل واحد من البيانات من ResultSet.

الدالة shutdown في السطر ١٣١ تقوم بإغلاق الاتصال مع قاعدة البيانات باستخدام الدالة close.

تسجيل قاعدة البيانات "الكتب" Books.mdb كمصدر بيانات في مصدر قواعد البيانات المفتوح ODBC.

إن المثال السابق يفترض بأن قاعدة البيانات المستخدمة مسجلة كمصدر بيانات ODBC، وما سنفعله الآن هو التعرف على كيفية تعريف قاعدة بيانات كمصدر بيانات ODBC. للقيام بهذا العمل علينا اتباع الخطوات التالية:

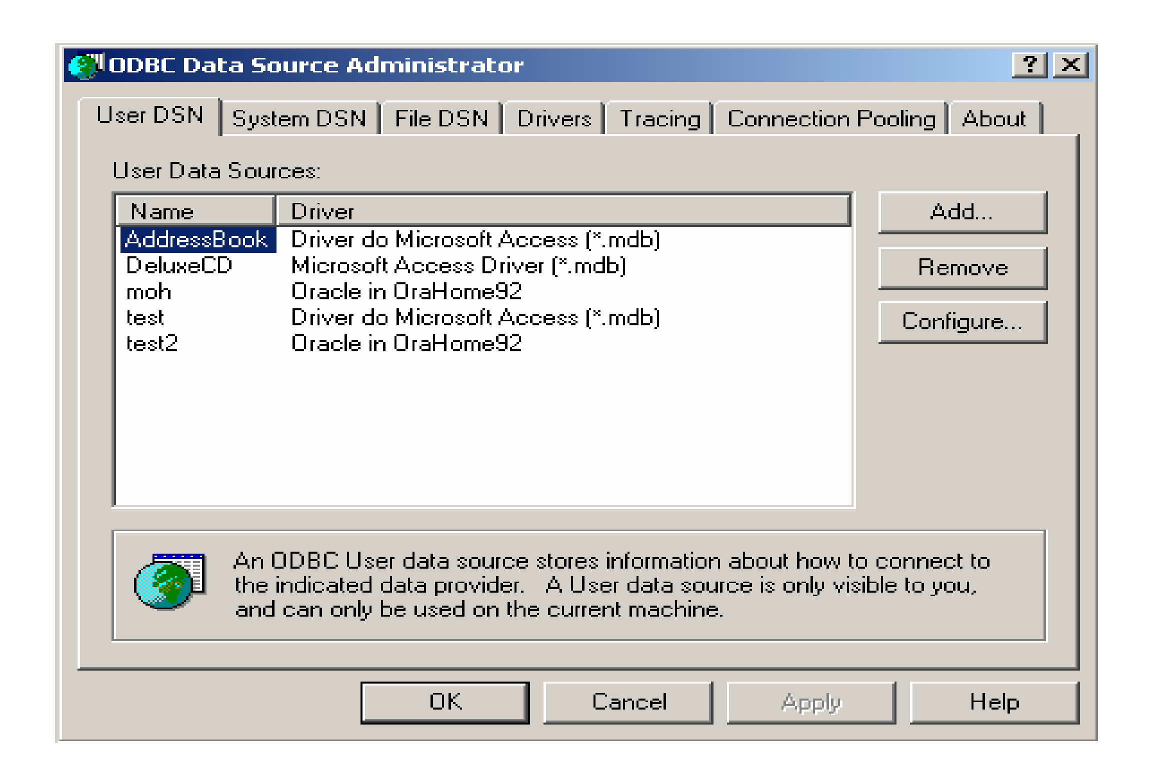
انتقل إلى Control Panel داخل نظام Windows

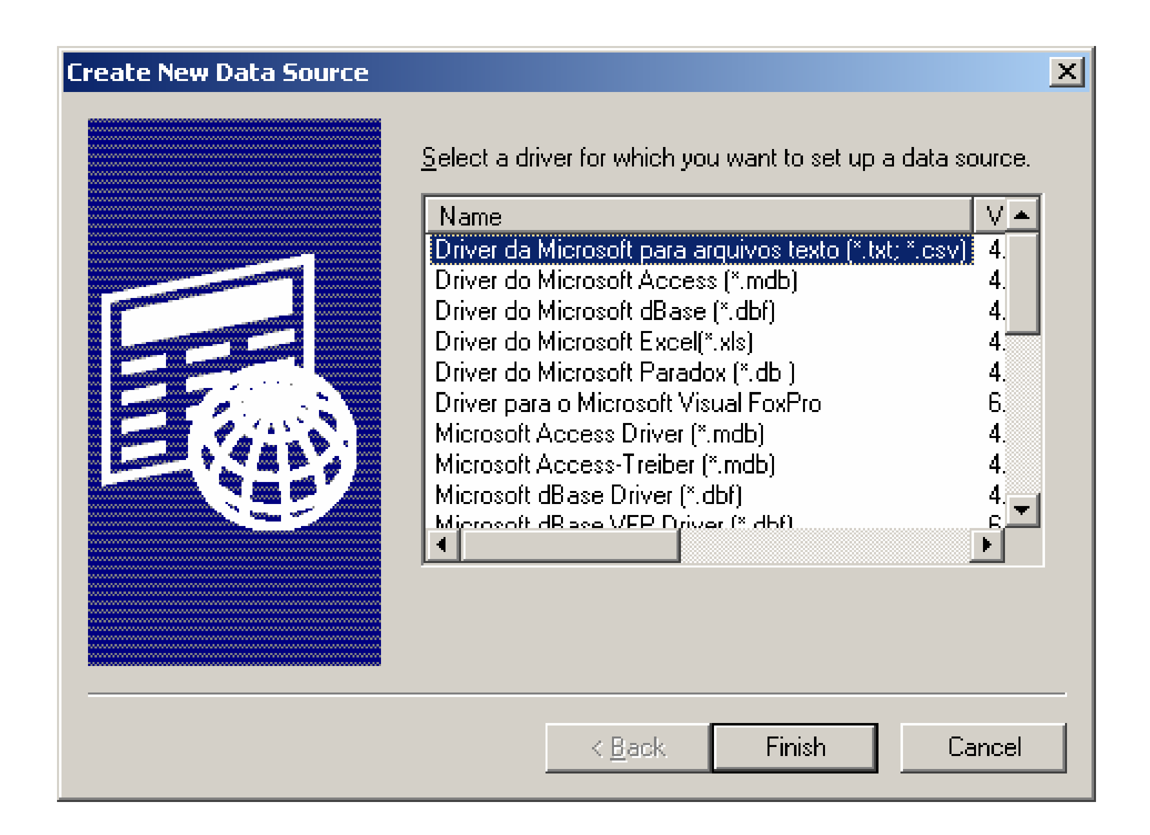
انقر نقرة مزدوجة على الخيار ODBC Data Sources، سيظهر لك الآن صندوق الحوار المبين ادناه.

في قائمة User DSN انقر على الزر Add لإظهار صندوق إنشاء مصدر بيانات جديد كما هو موضح بالشكل٢.

حيث إن قاعدة البيانات الخاصة بنا هي من نوع Access فسوف نستخدم Microsoft Access Driver.

انقر الزر إنهاء.





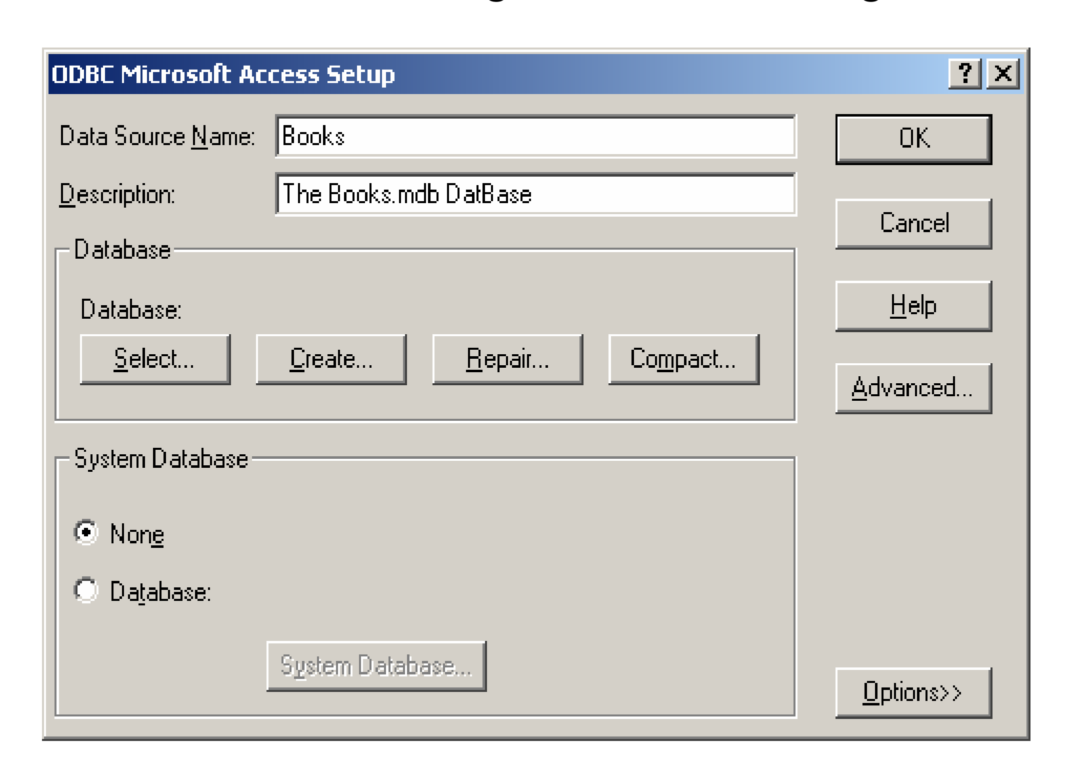
سيظهر لنا الآن صندوق الحوار الخاص بـ ODBC Microsoft Access، حيث سنقوم بتحديد كل مما يلي:

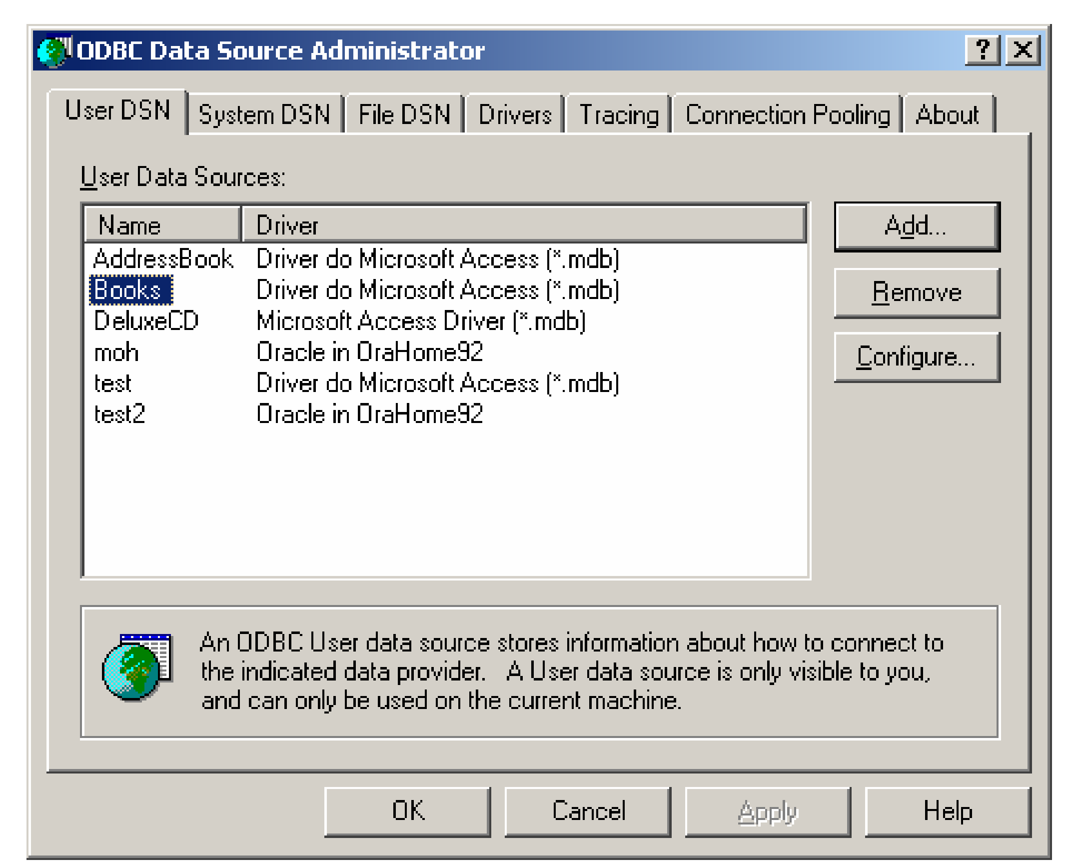
أدخل اسم قاعدة البيانات الذي سيستخدم من قبل JDBC للرجوع لقاعدة البيانات في الحقل Data Source Name.

يمكنك إدخال وصف لقاعدة البيانات (اختياري) في الحقل Description

١. انقر على زر Select
٢. ابحث ثم اختر اسم قاعدة البيانات الخاصة بك (في هذه الحالة نختار Books.mdb)
٣. انقر الزر OK
٤. انقر الزر Advanced لإظهار قائمة الخيارات المتقدمة
٥. أدخل اسم المستخدم anonymous
٦. أدخل كلمة المرور guest
٧. انقر الزر OK للخروج من صندوق الحوار
٨. ١١. انقر الزر OK مرة أخرى للخروج من ODBC Microsoft Access Setup
٩. ١٢. انقرر الزر OK مرة أخرى للخروج من ODBC Data Source Administrator

تستطيع الآن تنفيذ البرنامج المكتوب أعلاه لمشاهدة الناتج.





المثال الثاني:

في هذا المثال سنقوم بتعديل المثال الأول بحيث يستطيع المستخدم إدخال أي جملة استعلام، ثم يقوم البرنامج بتنفيذ هذه الجملة في قاعدة البيانات وعرض الناتج على الشاشة.

```
1	// DisplayQueryResults.java
2	// This program displays the ResultSet returned by a
3	// query on the Books database.
4	import java.sql.*;
5	import javax.swing.*;
6	import java.awt.*;
7	import java.awt.event.*;
8	import java.util.*;
9
10	public class DisplayQueryResults extends JFrame {
11	   // java.sql types needed for database processing
12	   private Connection connection;
13	   private Statement statement;
14	   private ResultSet resultSet;
15	   private ResultSetMetaData rsMetaData;
16
```

```
17        // javax.swing types needed for GUI
18        private JTable table;
19        private JTextArea inputQuery;
20        private JButton submitQuery;
21
22        public DisplayQueryResults()
23        {
24          super( "Enter Query. Click Submit to See Results." );
25
26          // The URL specifying the Books database to which
27          // this program connects using JDBC to connect to a
28          // Microsoft ODBC database.
29          String url = "jdbc:odbc:test";
30          String username = "moh";
31          String password = "moh";
32
33          // Load the driver to allow connection to the database
34          try {
35            Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
36
37            connection = DriverManager.getConnection(
38              url, username, password );
39          }
40          catch ( ClassNotFoundException cnfex ) {
41            System.err.println(
42              "Failed to load JDBC/ODBC driver." );
43            cnfex.printStackTrace();
44            System.exit( 1 );  // terminate program
45          }
46          catch ( SQLException sqlex ) {
47            System.err.println( "Unable to connect" );
48            sqlex.printStackTrace();
49            System.exit( 1 );  // terminate program
50          }
51
52          // If connected to database, set up GUI
53          inputQuery =
54            new JTextArea( "SELECT * FROM Authors", 4, 30 );
55          submitQuery = new JButton( "Submit query" );
56          submitQuery.addActionListener(
57            new ActionListener() {
58              public void actionPerformed( ActionEvent e )
```

```
59             {
60                getTable();
61             }
62          }
63       );
64
65       JPanel topPanel = new JPanel();
66       topPanel.setLayout( new BorderLayout() );
67       topPanel.add( new JScrollPane( inputQuery),
68                BorderLayout.CENTER );
69       topPanel.add( submitQuery, BorderLayout.SOUTH );
70
71       table = new JTable( 4, 4 );
72
73       Container c = getContentPane();
74       c.setLayout( new BorderLayout() );
75       c.add( topPanel, BorderLayout.NORTH );
76       c.add( table, BorderLayout.CENTER );
77
78       getTable();
79
80       setSize( 500, 500 );
81       show();
82    }
83
84    private void getTable()
85    {
86       try {
87          String query = inputQuery.getText();
88
89          statement = connection.createStatement();
90          resultSet = statement.executeQuery( query );
91          displayResultSet( resultSet );
92       }
93       catch ( SQLException sqlex ) {
94          sqlex.printStackTrace();
95       }
96    }
97
98    private void displayResultSet( ResultSet rs )
99       throws SQLException
100   {
```

```
101        // position to first record
102        boolean moreRecords = rs.next();
103
104        // If there are no records, display a message
105        if ( ! moreRecords ) {
106           JOptionPane.showMessageDialog( this,
107              "ResultSet contained no records" );
108           setTitle( "No records to display" );
109           return;
110        }
111
112        Vector columnHeads = new Vector();
113        Vector rows = new Vector();
114
115        try {
116           // get column heads
117           ResultSetMetaData rsmd = rs.getMetaData();
118
119           for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
120              columnHeads.addElement( rsmd.getColumnName( i ) );
121
122           // get row data
123           do {
124              rows.addElement( getNextRow( rs, rsmd ) );
125           } while ( rs.next() );
126
127           // display table with ResultSet contents
128           table = new JTable( rows, columnHeads );
129           JScrollPane scroller = new JScrollPane( table );
130           Container c = getContentPane();
131           c.remove( 1 );
132           c.add( scroller, BorderLayout.CENTER );
133           c.validate();
134        }
135        catch ( SQLException sqlex ) {
136           sqlex.printStackTrace();
137        }
138     }
139
140     private Vector getNextRow( ResultSet rs,
141                          ResultSetMetaData rsmd )
142         throws SQLException
```

```
143        {
144          Vector currentRow = new Vector();
145
146          for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
147            switch( rsmd.getColumnType( i ) ) {
148              case Types.VARCHAR:
149              case Types.LONGVARCHAR:
150                  currentRow.addElement( rs.getString( i ) );
151                break;
152              case Types.INTEGER:
153                  currentRow.addElement(
154                    new Long( rs.getLong( i ) ) );
155                break;
156              default:
157                System.out.println( "Type was: " +
158                  rsmd.getColumnTypeName( i ) );
159            }
160
161          return currentRow;
162        }
163
164        public void shutDown()
165        {
166          try {
167            connection.close();
168          }
169          catch ( SQLException sqlex ) {
170            System.err.println( "Unable to disconnect" );
171            sqlex.printStackTrace();
171          }
172        }
173
174        public static void main( String args[] )
175        {
176          final DisplayQueryResults app =
177            new DisplayQueryResults();
178
179          app.addWindowListener(
180            new WindowAdapter() {
181              public void windowClosing( WindowEvent e )
182              {
183                app.shutDown();
```

```
184                 System.exit( 0 );
185              }
186           }
187       );
188    }}
```

تمارين

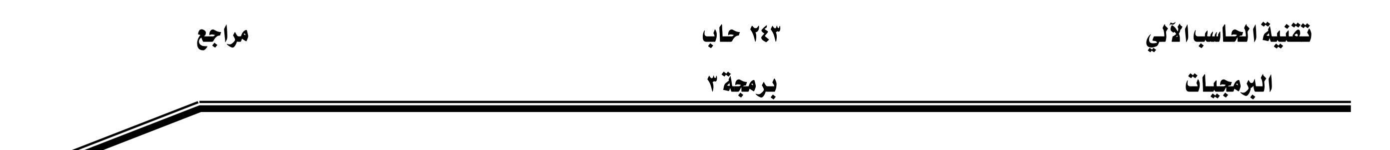
س١) أيها يوفر إمكانيات أكبر في معالجة البيانات: استخدام الملفات لتخزين البيانات أم الاتصال بقاعدة بيانات محددة، ولماذا؟

س٢) لقد رأيت كيف تعرف مشغل لربط قاعدة بيانات اكسس ببرنامج جافا. قم الآن بتعريف مشغل لربط قاعدة بيانات أوراكل ببرنامج جافا، وذلك باتباع نفس الخطوات الموضحة في الدرس مع اختيار قاعدة بيانات أوراكل بدلا من اكسس.

س٣)

١. عدل في المثال الثاني، المشروح في هذا الدرس بحيث يستخدم المشغل الذي قمت بتعريفه في السؤال الثاني.

٢. عدل واجهة التطبيق في نفس المثال لتتلائم مع الأعمدة المرتجعة من قاعدة البيانات أوراكل الخاصة بك.



المرجع

- Java How to Program, Deitel and Deitel, Fourth Edition

المحتويات

تقدر المؤسسة العامة للتعليم الفني والتدريب المهني الدعم

المالي المقدم من شركة بي آيه إي سيستمز (العمليات) المحدودة

GOTEVOT appreciates the financial support provided by BAE SYSTEMS

BAE SYSTEMS